图灵程序设计丛书

Peachpit Press

JavaScript Visual Quickstart Guide Ninth Edition



JavaScript 基础教程（第9版）

【美】Dori Smith Tom Negrino 著 陈建瓯 柳靖 等 译

经典JavaScript入门书最新版，通过实例透彻讲解Web开发相关技术。

原版累计销量200000册，中文版累计销量近50000册！

中国工信出版集团 人民邮电出版社
POSTS & TELECOM PRESS

Dori Smith

世界知名Web程序员和设计师，从事软件开发已有20多年。她是Web标准项目（WaSP）指导委员会委员，并发起成立了世界性的女性技术社区Wise-Women Web。自1995年开始，与Tom Negrino一起致力于向初学者介绍Web。

Tom Negrino

蜚声全球的技术作家，长期主持*Macworld*和许多其他技术杂志的专栏。自1995年开始，与Dori Smith合作著书，向初学者介绍Web，目前已著有数十本书。

图灵程序设计丛书

JavaScript Visual Quickstart Guide Ninth Edition

JavaScript基础教程

（第9版）

【美】Dori Smith　Tom Negrino 著　陈建瓯 柳靖 等 译

人民邮电出版社
北　京

图书在版编目（C I P）数据

JavaScript基础教程 : 第9版 / (美) 史密斯 (Smith,D.) , (美) 尼格瑞诺 (Negrino,T.) 著 ; 陈建瓯等译. -- 北京 : 人民邮电出版社, 2015.3
（图灵程序设计丛书）
ISBN 978-7-115-38522-2

Ⅰ. ①J… Ⅱ. ①史… ②尼… ③陈… Ⅲ. ①JAVA语言－程序设计－教材 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2015)第025437号

内 容 提 要

本书是经典的 JavaScript 入门书，以易学便查、图文并茂、循序渐进和善于用常见任务讲解语言知识而著称。书中从 JavaScript 语言基础开始，分别讨论了图像、框架、浏览器窗口、表单、正则表达式等内容，循序渐进地给出了 JavaScript 及相关的 CSS、DOM、Ajax 和 jQuery 等技术。第 9 版全新改写，新增更多示例和技术介绍，使用流行的 jQuery 框架向网站轻松添加有用的功能；更增添一章专门介绍如何为移动设备编写脚本。

本书适合有志于从事 Web 开发和设计的初学者，也是高等院校相关课程的理想入门教材。

◆ 著　　　　[美] Dori Smith　Tom Negrino
译　　　　陈剑瓯　柳　靖 等
责任编辑　朱　巍
责任印制　杨林杰
◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路11号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京　　　　印刷
◆ 开本：800×1000　1/16
印张：26
字数：709千字　　　　2015年 3 月第 1 版
印数：1－4 000册　　　　2015年 3 月北京第 1 次印刷
著作权合同登记号　图字：01-2014-7514号

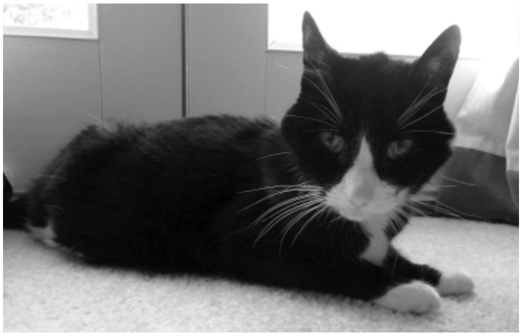
定价：69.00元
读者服务热线：(010)51095186转600　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京崇工商广字第 0021 号

版权声明

谨以本书纪念Bill Horwitz和Dorothy Negrino，他们饱含着求知的热情。

特别纪念

1997年，在写《JavaScript基础教程（第1版）》第1章的时候，我们费尽心思想找到某种方式说清楚JavaScript对象的概念，家里刚添的新成员Pixel猫给了我们灵感。这么多年来，无数读者告诉我们“猫对象”帮助他们更好地理解了JavaScript。Pixel成了我们许多书的吉祥物。2013年秋天，在跟我们愉快相伴多年后，Pixel因衰老去世了。如今这本书出版第9版，我们想告诉Pixel：大家都很想念你。



Pixel最后一次跟我们在一起

前 言

欢迎使用JavaScript！使用这种容易上手的程序设计语言，可以给网页增色，使网页更好用。本书是一本轻松的JavaScript入门教程，所以即使不是计算机高手，你也可以由此很快学会脚本编写。任何时候你都不需要借助工具包。就像我们的一位朋友说的："我们已经够疯狂了，所以你没有必要这样做啦！"

读者对象

我们猜想既然你有兴趣学习JavaScript，那么肯定有创建HTML页面和Web站点的经验，而且希望更进一步，使站点更具交互性。学习本书并不要求你了解任何编程或脚本编程的知识，也不要求你是HTML专家（当然，如果你是HTML专家也无妨）。我们只假设你具备构建网页的基本知识，而且熟悉常用的HTML标签，比如链接、图像和表单。同样，我们也希望你了解现代网页另一个主要构件块——CSS的基础知识。

在某些章题为"目前需要了解的HTML知识"的表中，我们对HTML做了一些解释。并非每章都有这部分内容，只有在我们认为你需要速查的地方才会提供。有了这些HTML信息，你就不需要在阅读本书的同时，再去翻另一本书或者打开相关网页查找HTML属性的语法了[①]。

如果你对编程有所了解，应该会注意到本书介绍JavaScript的方式与其他图书的风格很不一样。本书并不深入介绍JavaScript的语法和结构，而且本书也不是一本深入而全面的语言参考书（当然附录A中提供了一些很有价值的内容）。这方面市面上已经有不少很好的书，附录D中列了出来。本书和这些书之间的差异是，本书并不拘泥于形式，而是集中地演示如何用JavaScript完成一些有用的任务，不赘述大量的额外信息。

在本书的前几版中，我们增加了对Ajax和jQuery的介绍。这种技术结合使用了JavaScript和其他常用的Web技术为网页增加交互性，并且改善了Web站点的用户体验。在这个版本中，我们添加了更多示例和技术介绍，使用流行的jQuery框架向网站轻松添加有用的功能。

如何使用本书

本书采用一些特殊的版式，帮助你更轻松地学习和理解JavaScript。

组成本书的大部分内容由分步说明构成。我们在书中以特殊的字体样式表示HTML或JavaScript代码，比如：

① 要学习HTML，推荐阅读《HTML5与CSS3基础教程（第8版）》（http://www.ituring.com.cn/book/1199）。——编者注

```
<div id = "thisDiv"> window.onload = initLinks;
```

你还会注意到，HTML和JavaScript代码都显示为小写。这么做是因为，这个版本中的所有脚本都符合W3C（World Wide Web Consortium，万维网联盟）的HTML5标准。当在JavaScript中看到引号时，总是直引号（'或"），而不是弯引号（‘或“）。弯引号会使JavaScript失效，在编写脚本时应该避免使用。

在与分步说明对应的脚本中，我们以粗体突出显示脚本中正在讨论的部分，这样你就能够马上找到我们正在讨论的代码。我们还常常在Web浏览器窗口的屏幕图中以灰色底纹突出显示其中某些重要的部分。

因为图书的页面比计算机屏幕窄，所以一些JavaScript代码行在页面上排不开。出现这种情况时，我们将代码行分为多行，在接续行前面使用箭头→表示这是续行，并且将续行缩进，如下所示：

```
dtString = "Hey, just what are you
→doing up so late?";
```

关于浏览器

从第6版起有一个大的变化：我们不再支持那些版本非常老的浏览器或者那些在支持Web标准方面做得很差的浏览器。我们发现，几乎所有Web用户都升级到了现代浏览器，它们充分地支持公认的Web标准（比如HTML、CSS2和DOM）。这包括IE 9或更高版本，Firefox、Safari和Chrome的所有版本，以及Opera 7或更高版本。

我们在几种操作系统上的多个浏览器中测试了脚本，这些操作系统包括Windows（主要是Windows 7，有少数情况是Windows 8，不再支持Windows XP和Vista）、OS X（10.8.5或更高版本）和Ubuntu Linux（只在Ubuntu的默认浏览器Firefox中测试了脚本）。

我们使用最主流的浏览器——微软Windows版本的IE——虚拟测试了本书中的所有内容（使用了IE 9、IE 10和IE 11）。这一版，我们还增加了针对Mac和Windows平台上持续更新的谷歌Chrome的测试。我们还用Mac和Windows平台上最新版本的Firefox（每几周就更新一次，本书使用的最新版本为29）对脚本进行了测试。此外，还使用Mac平台上的Safari 6和7进行了测试。这意味着这些脚本对于从WebKit引擎衍生出来的任何浏览器都可以正常使用，对基于KHTML（Safari最开始使用的开源呈现引擎）的浏览器（比如Linux浏览器Konqueror）也可以正常使用。WebKit也是移动操作系统浏览器核心之一，比如苹果的iOS、谷歌的Android、亚马逊的Kindle Fire平板电脑，以及黑莓的Blackberry 10。对于移动设备，我们主要将脚本在iPhone和iPad上面进行了测试。

不必输入代码

一些JavaScript图书只在书中印刷出脚本代码，你在实践时必须自己输入代码。我们认为这种方式已经过时了。作者们不得不完成这些艰苦的输入工作，但是你不必重复这些劳动。我们为本书提供了一个配套Web站点，其中包含本书中的所有脚本，你可以将这些脚本复制并粘贴到自己的网页中。如果我们在书中发现了任何错误，也会在这里更正。这个配套站点的网址是www.javascriptworld.com。

如果由于某种原因你打算输入某些脚本示例，那么可能会发现这些示例似乎不起作用，这是因为你没有这些示例所用的支持文件。例如，在图像上实现屏幕效果的示例中，需要图像文件。但请放心，

这些文件都放在本书的Web站点上了，而且打包好了供你下载。你找到的可下载文件包含所有脚本、HTML文件、CSS文件以及用到的所有媒体文件。如果你遇到任何问题，可以查看配套Web站点上的FAQ（常见问题）。

如果阅读了FAQ，而你的问题还没有解决，可以通过js9@javascriptworld.com给我们发邮件。很抱歉地说一声，因为收到的邮件太多，所以我们不可能也不会回复那些把本书问题发送到我们个人邮箱的邮件。但是，我们可以保证发送到js9@javascriptworld.com的邮件会得到答复。

当然，自己输入代码可能会帮你更彻底地理解JavaScript，那么但做无妨！

开始吧

JavaScript最好的一点是它很容易用一个简单的脚本在网页上实现很酷的效果，然后根据需要逐渐添加更复杂的素材。你不必等到学完整本书，就可以开始改进自己的网页。等到你看完本书的时候，将能用Ajax和jQuery给站点添加高级的交互效果了。

当然，千里之行，始于足下。欢迎光临，请勿将手伸出窗外，照相时请不要用闪光灯。探索JavaScript的旅程已经开始。

致谢

感谢我们的编辑Nancy Peterson，她的专业精神、冷静沉着和温暖的关怀使本书的写作过程非常愉悦。特别感谢因我们的私事耽误出版流程时她的额外付出。

同样感谢本书另一名编辑Scholle McFarland，在Nancy工作安排超负的情况下，他的加入确保了本书的顺利出版。

感谢Scout Festa熟练的文字加工工作。衷心感谢本书的排版人员Danielle Foster，她优雅镇定地完成了排版工作。感谢给本书编索引的Emily Glossbrenner，她做了一项吃力不讨好的工作。

感谢Peachpit出版社的Nancy Ruenzel和Nancy Davis给予支持。

感谢采用本书以前的版本作为课程教材的所有高中、大中专院校和大学的教师。

目　　录

第 1 章

了解 JavaScript

1

对于Web站点的开发者来说,HTML的发展是一件好坏参半的事。在万维网发展的早期,HTML相当简单，很容易就能够掌握设计网页所需知道的一切。

随着 Web 的发展，页面设计人员还希望他们的页面能够与用户进行交互，HTML 很快就显得不足以满足这一需求了。Netscape 发明了 JavaScript，作为控制浏览器和给网页添加活力和交互性的方法。

自从诞生以来，JavaScript 已经经历了不小的演化，有时候在不同的浏览器上演化的方向有所不同。在本书后面，我们将详细讨论 JavaScript 的演化。

在本章中，你将了解 JavaScript 是什么（以及不是什么），它可以做什么（以及不能做什么）和 JavaScript 语言的一些基础知识。另外，还会向你介绍 Ajax，这是 JavaScript 和其他技术的一种组合，它在 Web 站点的交互性和创造性方面掀起了新的浪潮。

1.1 JavaScript 是什么

JavaScript 是一种可以用来给网页增加交互性的编程语言。但是，如果你不是程序员，那么也不必担心。Web 上有大量 JavaScript 代码，复制一下并稍作修改，就可以供自己使用。实际上，这种“站在其他程序员肩膀上”的方式正是熟悉 JavaScript 的好方法。

为了帮助你熟悉 JavaScript，我们建立了一个与本书配套的 Web 站点。在这个站点上提供了本书中的所有脚本（这样，你就不用自己输入了），以及更多的说明、附加资料和更新内容。站点的网址是 www.javascriptworld.com。

常常会看到 JavaScript 被称为“脚本语言”（scripting language），这暗示着它更适合编写脚本而不是程序。这实际上并没有根本性的差异。JavaScript 脚本也是一种程序，它们包含在 HTML 页面内部（原先编写脚本的方式），或者驻留在外部文件中（现在的首选方法）。在 HTML 页面上，因为脚本文本包围在<script>标签中，所以它不会显示在用户的屏幕上，而 Web 浏览器知道应该运行 JavaScript 程序。<script>标签常常放在 HTML 页面的<head>部分中，如脚本 1-1 所示。但是如果愿意，也可以将脚本放在<body>部分中。

脚本 1-1 这个非常简单的脚本在浏览器窗口中输出“Hello, Cleveland!”

```
<!DOCTYPE html>
<html>
```

```
<head>
     <title>Barely a script at all</title>
     <script type="text/javascript">
        window.onload = function() {
           document.getElementById ("myMessage").innerHTML = "Hello, Cleveland!";
        }
     </script>
</head>
<body>
<h1 id="myMessage">
</h1>
</body>
</html>
```

如果你还不熟悉这些 HTML 概念，需要关于 HTML 的更多信息，那么我们建议你读一下 Elizabeth Castro 和 Bruce Hyslop 所著的《HTML5 与 CSS3 基础教程（第 8 版）》。如果你只需要简单回顾一下，那么本书的许多章节都提供了“目前需要了解的 HTML 知识”，其中列出了相关的 HTML 标签。

1.2 JavaScript 不是 Java

尽管名字中有 Java，但是 JavaScript 和 Java 之间没有太大的关系。Java 是一种功能完备的编程语言，由 Sun 公司开发，由 Oracle 公司推广（自从 Oracle 收购 Sun 公司以来）。Java 是 C 和 C++编程语言之后的又一种主流语言，程序员可以使用它创建完整的应用程序和控制消费类电子设备。与其他语言不同，Java 宣称具有跨平台兼容性。也就是说，程序员应该能够编写出可以在所有种类的机器上运行的 Java 程序，无论机器运行的是 Windows、Mac OS X 还是任何风格的 Unix。但实际上，Java 不总是能够实现这个梦想，因为 Sun 公司和微软公司在这种语言的发展方向方面有很大的分歧。微软公司首先试图以自己的方式将 Java 集成到 Windows 中（Sun 公司认为，这种方式会使 Java 在 Windows 上以一种方式工作，而在其他机器上以另一种方式工作，从而破坏了 Java 的跨平台兼容性）。随后，微软公司从 Windows 中完全去除了 Sun 公司的 Java，而创建了自己的类 Java 语言：C#。经过两公司之间的一轮诉讼之后，Sun 公司占据了上风，微软从 Windows 中移除了自己的 Java。现在，不论是哪种平台，都可以安装最新的 Java 版本（www.java.com/getjava/）。

除了单独的应用程序之外，Java 主要用于在*客户端*（client side，即用户的浏览器中）创建 applet。applet 是一种通过因特网下载并在 Web 浏览器中运行的小程序。因为 Java 具有跨平台性质，所以这些 applet 应该能够在任何支持 Java 的浏览器中以相同的方式运行。在近几年中，我们看到许多 Java applet 被 Adobe Flash 动画替代了，因为一般来说 Adobe Flash 动画比 Java applet 更容易创建。近几年来，随着计算处理速度和浏览器中 JavaScript 功能的不断提升，在客户端使用 Java（和 Flash）的情况越来越少。然而，Java 已经成为一种在服务器端编写代码的流行语言。

可以使用`<object>` HTML 标签将 Java applet 嵌入网页，还要提供指定 applet 的附加信息。当浏览器看到`<object>`标签时，它会从服务器下载 Java applet，然后 applet 就会在这个标签中指定的屏幕区域中运行（见图 1-1）。

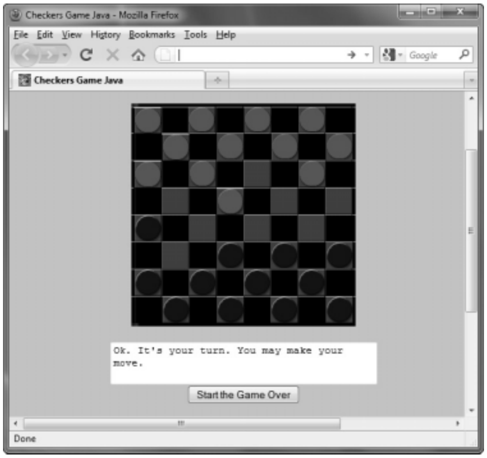


图 1-1　这个 Java applet 实现了一种西洋跳棋游戏

1.3　JavaScript 的起源

既然 JavaScript 与 Java 无关，那么为什么它们的名称如此相似呢？这是计算机行业最烦人的恶习之一：为了获得市场营销方面的成功，而不顾及产品实质。

当 Netscape 在其 Navigator Web 浏览器中添加了一些基本脚本功能时，它最初将这种脚本语言称为 LiveScript。与此同时，Java 开始大行其道，它被认为是计算行业中下一项伟大的革新。当 Netscape 在 Navigator 2 中支持运行 Java applet 时，它也将 LiveScript 改名为 JavaScript，希望以此借用 Java 的声势。尽管 JavaScript 和 Java 是非常不同的编程语言，但这一事实并没有阻止 Netscape 采用这种市场营销手段。从那时候开始，我们这些技术作家只好不厌其烦地解释 JavaScript 和 Java 是不同的东西。当然，我们靠这挣了不少钱，从这个角度来说，我们可能应该感谢这些市场营销专家。

当微软公司看到 JavaScript 在 Web 开发人员中流行起来时，它意识到必须在 IE 中添加一些脚本功能。它原本可以采用 JavaScript，但是与通常情况一样，微软公司又自行其事，建立了自己的脚本语言，这种语言非常像 JavaScript，但又不完全相同。JavaScript 的微软版本称为 JScript。

1.4　JavaScript 可以做什么

用 JavaScript 可以做许多事情，使网页更具交互性，给站点的用户提供更好、更令人兴奋的体验。JavaScript 使你可以创建活跃的用户界面，当用户在页面间导航时向他们提供反馈。例如，你可能在一些站点上见过，当鼠标指针停留在按钮上时，会突出显示按钮。这是用 JavaScript 实现的，使用了一种称为翻转器（rollover）的技术（见图 1-2）。



图 1-2　翻转器是一个当鼠标指针停留在其上时会改变的图像

可以使用 JavaScript 来确保用户以表单形式输入有效的信息，这可以节省你的业务时间和开支。如果表单需要进行计算，那么可以在用户机器上用 JavaScript 来完成，而不需要任何服务器端处理。你应该知道一种区分程序的方式：在用户机器上运行的程序称为*客户端*（client-side）程序，在服务器上运行的程序称为*服务器端*（server-side）程序。

使用 JavaScript，根据用户的操作可以创建自定义的 Web 页面。假设你正在运行一个旅行指南站点，用户单击夏威夷作为旅游目的地。你可以在一个新窗口中显示最新的夏威夷旅游指南。JavaScript 可以控制浏览器，所以你可以打开新窗口、显示警告框以及在浏览器窗口的状态栏中显示自定义的消息。因为 JavaScript 有一套日期和时间特性，可以生成时钟、日历和时间戳文档。

JavaScript 还可以处理表单，设置 cookie，即时构建 HTML 页面以及创建基于 Web 的应用程序。

1.5　JavaScript 不能做什么

JavaScript 是一种*客户端*语言[①]。也就是说，设计它的目的是在用户的机器上而不是服务器上执行任务。因此，JavaScript 有一些固有的限制，这些限制主要出于如下安全原因。

- ❑ JavaScript 不允许写服务器机器上的文件。尽管写服务器上的文件在许多方面是很方便的（比如存储页面单击数或用户填写的表单数据），但是 JavaScript 不允许这么做。而是需要用服务器上的一个程序处理和存储这些数据，这个程序可以是用 Java、Perl 或 PHP 等语言编写的。
- ❑ JavaScript 不能关闭不是由它自己打开的窗口。这是为了避免一个站点关闭其他任何站点的窗口，从而独占浏览器。
- ❑ JavaScript 不能从来自另一个服务器的已经打开的网页中读取信息。换句话说，网页不能读取已经打开的其他窗口中的信息，因此无法探察访问这个站点的冲浪者还在访问其他哪些站点。

1.6　JavaScript 及其他

1.6.1　jQuery是什么

待你多为几个项目编写点 JavaScript 代码后，你很可能会发现自己总会反复不断地编写某些一样的东西——例如，针对不同浏览器的解决方案。如果你觉得应该存在更好的方式处理这些常用的代码段，那么你的感觉非常正确——很多程序员的想法跟你一样。

为了更方便地针对已解决的问题套用已编写好的代码，程序员将他们的代码段捆绑到所谓的工具包、库或者框架中。这类东西很多，jQuery 是其中最常用的一种（jquery.com）。

说到这里你可能已经猜到了，jQuery 本身并不是一种语言，而是一个实用程序和控件集，我们可以将它添加到 JavaScript 中，帮助我们用更少量的代码编写更高级的站点。

本书第 14 ~ 16 章会详细讲解 jQuery。

1.6.2　Ajax是什么

简单的回答是，Ajax 是一种创建交互式 Web 应用程序的方式。这究竟是什么意思呢？我们来考

① 实际上，也存在服务器端实现的 JavaScript 版本，如 Rhino。——编者注

虑一下可能通过 Web 站点做的事情。例如，你可能想和自己的爱人出去吃饭（没有爱人的可能要和自己的朋友、亲戚或宠物一起去吃饭。大部分人不会和宠物出去吃饭，我们就这么一说）。所以，你希望知道如何从当前的地点到达市内大家都在谈论的那家有名的新餐馆。你决定到一个提供地图的 Web 站点上查找路线。你会进入这个站点，输入餐馆的地址，站点会显示一张标出了这个餐馆的地图。这个站点会显示带边框的地图，如果希望改变地图的视域，那么可以单击边框。单击边框，等大约 5 ~ 10 秒，地图就会重新绘制，如果希望再次改变视域，就重复操作。这个过程很慢，而且响应性很差。如果只是单击地图并向希望的方向拖动它，地图视域就会随着鼠标拖动而移动，那不是更好吗？

可以通过用 Ajax 构建的 Web 应用程序向用户提供这种动态的交互性（见图 1-3）。这样的话，用户几乎不需要等待，并一直有控制能力，而且可以创建具有与传统桌面应用程序相同用户体验的基于 Web 的应用程序。这样，用户就能够更快、更轻松地查明如何从家到达那家有名的餐馆。

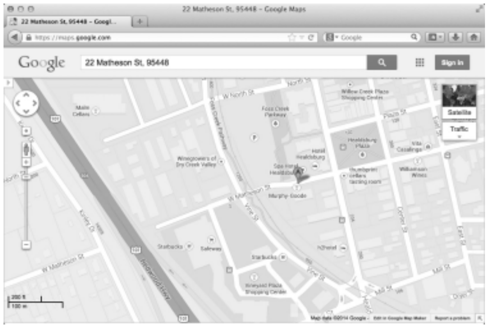


图 1-3 支持 Ajax 的 Google Maps 可以提供更流畅更具交互性的用户体验

Ajax 是 Asynchronous JavaScript and XML（异步 JavaScript 和 XML）的缩写，这个词是由 Web 开发人员 Jesse James Garrett 在 2005 年年初首创的。严格地说，Ajax 只是 JavaScript 的一小部分（只是这一部分特别流行）。但是，随着频繁地使用，这个词不再指某种技术本身（比如 Java 或 JavaScript）。

在大多数情况下，Ajax 一般是指以下这些技术的组合：

- HTML；
- CSS（Cascading Style Sheet，层叠样式表）；
- 使用 JavaScript 访问的 DOM（Document Object Model，文档对象模型）；
- XML 或 JSON，这是在服务器和客户端之间传输的数据格式；
- XMLHttpRequest，用来从服务器获取数据。

这个列表有点儿复杂，尤其是对于在 JavaScript 或其他 Web 编程方面经验不太丰富的人。但是，不必担心，我们在本书中会介绍这些技术。在学到关于 Ajax 的章节时，你应该已经掌握了组成 Ajax 的各种技术。

Ajax 的好处是，应用程序的大多数处理在用户的浏览器中发生，而且对服务器的数据请求往往很短。所以可以使用 Ajax 建立功能丰富的应用程序，这些应用程序依赖基于 Web 的数据，但是其性能远远超过老式方法，因为老式方法要求服务器传回整个 HTML 页面来响应用户操作。

一些公司已经在 Ajax 方面投入大量资金，尤其是谷歌。谷歌已经建立了几个著名的 Ajax 应用程序，包括 Gmail（基于 Web 的电子邮件）、Google Calendar、Google Docs 和 Google Maps。另一个大型的 Ajax 支持者是 Yahoo!，它使用 Ajax 增强个性化的 My Yahoo!门户、Yahoo!首页、Yahoo! Mail，等等。这两家公司都向公众开放了其 Web 应用程序的接口，人们可以使用这些接口建立有意思的新应用程序。例如，许多人为 Google Maps 创建了 mashup（混搭），这些程序会获得地图并在地图上加上有意思、有用或好玩的信息，比如洛杉矶地区所有日本餐馆的位置或电影摄影棚的位置。

第 13 章会更详细地介绍 Ajax。

✔提示

- ❑ 在 Google Maps Mania（googlemapsmania.blogspot.com）上可以找到许多 Google Maps mashup 的列表。

1.7　组合式语言

还有一点我们应该注意：JavaScript 是一种面向对象（object-oriented）的语言。这意味着什么呢？

1.7.1　对象

首先，我们来考虑对象。对象（object）就是某种东西。在现实中，一只猫、一台计算机和一辆自行车都是对象（见图 1-4）。对于 JavaScript，它处理的对象都在 Web 浏览器中，比如窗口、表单，以及按钮和复选框等表单元素（见图 1-5）。

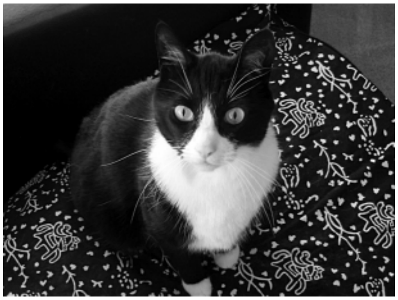


图 1-4　猫对象（这只猫名叫 Pixel）



图 1-5　按钮和复选框是浏览器对象，可以用 JavaScript 来操纵

因为你可以有多只猫或者多个窗口，所以给它们起名字是有意义的。你可以把自己的宠物叫作一号猫和二号猫，但是这不是一种好想法，原因有两个：首先，如果它们有唯一的名称，就更容易区分它们；其次，这有点不礼貌。因此，本书中的所有示例将给对象起唯一的名字。

✔提示

- ❑ 在因特网上可能会看到一些脚本用 `window[0]`和 `form[1]`这样的名称来称呼对象。由于以上原因，这种方式并不好。如果给脚本中的不同对象起名字，而不是使用数字，那么跟踪对象会容易得多。
- ❑ 有些爱挑剔的程序员会认为，JavaScript 不是面向对象的，而是基于对象（object-based）的。在本书中，这两个意思相同。

1.7.2　属性

对象具有属性（property）。猫有毛皮，计算机有键盘，自行车有轮子。在 JavaScript 环境中，文档有标题，表单可以有复选框。

改变对象的属性就修改了这个对象。相同的属性名可以用于完全不同的对象。假设有一个名为 `empty` 的属性。在任何合适的地方都可以使用 `empty`，所以可以说猫的肚子空了，也可以说猫的食盆空了。

注意，计算机的键盘和自行车的轮子不仅仅是属性，它们本身也是对象，可以具有自己的属性。所以，对象可以有子对象。

1.7.3　方法

对象可以做的事情称为*方法*（method）。猫会叫，计算机会崩溃，自行车可以前进。JavaScript 对象也有方法：按钮的 `click()`，窗口的 `open()`，文本的 `selected()`。圆括号表示它们是方法，而不是属性。

✔提示

- ❑ 可以把对象和属性看作名词，把方法看作动词。前者是东西，后者是这些东西可以完成的活动或对它们执行的操作。

1.7.4　将对象、属性和方法组合在一起

可以将对象、属性和方法组合在一起，从而更好地描述对象或过程。在 JavaScript 中，这些成分由点号分隔（就像因特网地址中的那样）。这称为*点号语法*（dot syntax）。下面是按这种方式编写的对象及其属性的一些示例：

```
bicycle.wheels
cat.paws.front.left
computer.drive.dvd
document.images.name
window.status
```

下面是按照点号语法编写的对象及其方法的一些示例：

```
cat.purr()
document.write()
forms.elements.radio.click()
```

1.7.5　DOM简介

在网页上，组成页面（或*文档*）的对象被组织在一个树型结构中。在以前构建 HTML 页面的时候，你已经知道这种结构了。页面的顶级包含在<html>标签中，在其中会找到<head>和<body>标签，而其他标签包含在这两个标签中，依次类推。某些浏览器可以显示这种树型结构，如图 1-6 所示。JavaScript 将文档树中的每一项都当作对象，可以使用 JavaScript 操纵这些对象。用来表示文档中对象的标准模型就称为 DOM（Document Object Model，文档对象模型）。

树中的每个对象也称为树的*节点*（node）。可以使用 JavaScript 修改树的任何方面，包括添加、访问、修改和删除树上的节点。树上的每个对象是一个节

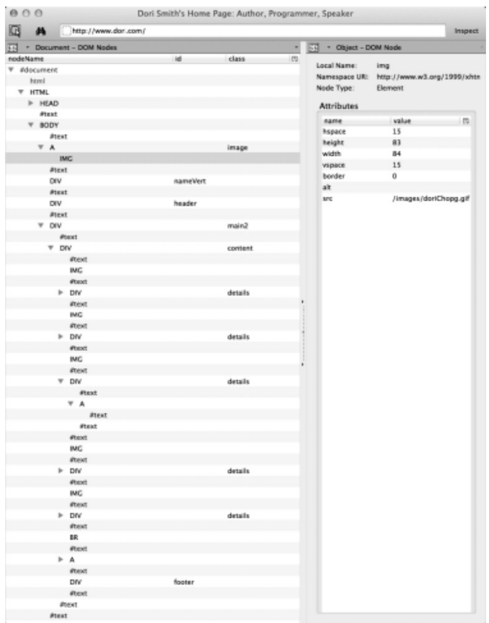


图 1-6　可以使用 DOM Inspector（Firefox 的一个插件）查看文档的树型结构。在 Safari、Chrome 和 IE 中也有相似的特性

点。如果节点包含 HTML 标签，那么它就称为元素节点（element node），否则，就称为文本节点（text node），当然，元素节点可以包含文本节点。这就是目前关于 DOM 和节点需要知道的所有知识，在本书中（尤其是第 10 章）会看到关于它们的更多信息。

1.8　处理事件

事件（event）是用户在访问页面时执行的操作。提交表单和在图像上移动鼠标就是两种事件。

JavaScript 使用称为事件处理程序（event handler）的命令来处理事件。用户在页面上的操作会触发脚本中的事件处理程序。表 1-1 列出了最常用的 12 种 JavaScript 事件处理程序。在第 8 章中，我们会讨论其他更高级的事件处理程序。

表1-1　事件处理程序

事　件	处理什么
`onabort`	用户终止了页面的加载
`onblur`	用户离开了对象
`onchange`	用户修改了对象
`onclick`	用户单击了对象
`onerror`	脚本遇到了一个错误
`onfocus`	用户激活了对象
`onload`	对象完成了加载
`onmouseover`	鼠标指针移动到对象上
`onmouseout`	鼠标指针离开了对象
`onselect`	用户选择了对象的内容
`onsubmit`	用户提交了表单
`onunload`	用户离开了页面

例如，猫就可以通过执行 `purr`（打呼噜）和 `stretch`（伸懒腰）操作来处理 `onpetting`（亲热）事件。

在 JavaScript 中，如果用户单击了一个按钮，那么 `onclick` 事件处理程序会注意到这一操作，并执行分配给它的任务。

在编写脚本时，不必预测出用户可能采取的所有操作，只需处理那些你希望提供特殊处理的事件。例如，如果没有 `onload` 事件处理程序，页面也会顺利地加载。但是，如果希望在加载页面时触发一个脚本，就要使用 `onload` 命令。

1.9　值和变量

在 JavaScript 中，一段信息就是一个值（value）。值有不同的类型，大家最熟悉的类型是数字。字符串（string）值是包围在引号中的一个或多个单词。表 1-2 列出了 JavaScript 值的其他类型。

表1-2　值类型

类　型	描　述	示　例
数字	任何数字值	3.141592654
字符串	引号中的字符	"Hello, world!"
布尔值（Boolean）	true或false	true
空值（null）	空且无含义	
对象	与对象相关联的任何值	
函数	函数返回的值	

变量（variable）是用来保存值的。例如，变量 myName 被赋值为字符串"Dori"。编写这一赋值的一种方式是 myName="Dori"。等号可以读作“设置为”。换句话说，变量 myName 现在包含值"Dori"。

✔提示

- ❑ JavaScript 是区分大小写的。这意味着 myname 与 myName 并不相同，也与 MyName 不相同。
- ❑ 变量名不能包含空格或其他标点符号，也不能以数字开头。它们还不能是 JavaScript 保留字。附录 B 列出了 JavaScript 保留字。

1.9.1　操作符

操作符（operator）是用来操作变量的符号。你应该已经熟悉简单算术中的操作符，加号和减号就是操作符。表 1-3 列出了大多数常见的操作符。

表1-3　操作符

操　作　符	作　用
x + y（数字）	将x和y相加
x + y（字符串）	将x和y拼接在一起
x - y	从x中减去y
x * y	将x和y相乘
x / y	将x除以y
x % y	x和y的模（即x除以y的余数）
x++、++x	给x加1（相当于x = x + 1）
x--、--x	给x减1（相当于x = x - 1）
-x	x的相反数

✔提示

- ❑ x++和++x 都是给 x 加 1，但是它们并不相同。前者在完成赋值之后再将 x 加 1，而后者将 x 加 1 之后再赋值。例如，如果 x 是 5，y=x++会将 y 设置为 5，x 设置为 6；而 y=++x 会将 x 和 y 都设置为 6。递减操作符--的工作方式相似。
- ❑ 如果数字和字符串相加，那么结果是一个字符串。例如，"cat"+5 的结果是"cat5"。

1.9.2　赋值和比较

在将一个值放进变量中时，就是将这个值赋给这个变量，这个任务要使用赋值操作符来完成。例

如，使用等号操作符进行赋值，比如 hisName="Tom"。表 1-4 列出了所有的赋值操作符。

表1-4　赋值

赋　　值	作　　用
x = y	将x设置为y的值
x += y	相当于x = x + y
x -= y	相当于x = x - y
x *= y	相当于x = x * y
x /= y	相当于x = x / y
x %= y	相当于x = x % y

除等号之外，其他赋值操作符都是用来修改变量值的简写形式。例如，x=x+5 的简写形式是 x+=5。为清楚起见，本书中的大多数地方都使用非简写形式。

1.9.3　比较

常常需要对两个变量的值进行比较，或者将一个变量的值与一个字面值（即输入表达式中的值）进行比较。例如，可能希望将周中日的值与"Tuesday"进行比较，这可以通过检查 todaysDate=="Tuesday"来完成。比较操作符的完整列表见表 1-5。

表1-5　比较

比　　较	作　　用
x == y	如果x和y相等，那么返回true
x === y	如果x和y完全相同，那么返回true
x != y	如果x和y不等，那么返回true
x !== y	如果x和y不完全相同，那么返回true
x > y	如果x大于y，那么返回true
x >= y	如果x大于等于y，那么返回true
x < y	如果x小于y，那么返回true
x <= y	如果x小于等于y，那么返回true
x && y	如果x和y都是true，那么返回true
x || y	如果x或y之一是true，那么返回true
!x	如果x是false，那么返回true

✔提示

- 如果对字符串进行比较，那么要知道"a"大于"A"，"abracadabra"小于"be"。

1.10　编写对 JavaScript 友好的 HTML

因为将使用 JavaScript 操纵文档中的对象，所以希望以适当的方式编写 HTML，使脚本能够轻松地处理 HTML。这基本上意味着要编写现代的符合标准的 HTML，并使用 CSS 将文档的结构与它的表现分隔开。

我们说“现代的 HTML”的意思不仅仅指使用 validator.w3.org 上的 Web 工具进行 W3C 检验，还应该提前考虑可能对页面进行什么操作，并添加适当的标签和属性，使 JavaScript 能够轻松地访问对象。需要哪些标记呢？很高兴你这么问。

1.10.1　结构、表现和行为

CSS 是一种用于 Web 的标准布局语言，可以控制版面、颜色以及元素和图像的大小和位置。HTML 文档应该利用外部样式表来定义文档中使用的样式。JavaScript 也应该放在外部文档中，这个文档应该只包含 JavaScript 代码。

按照这种方式进行分离，站点将包含以下 3 种文本文件。

- HTML：包含页面的内容和结构。
- CSS：控制页面的外观和表现。
- JavaScript：控制页面的行为。

如果这么做，对站点进行修改就会很容易，甚至修改全站点范围的效果也很容易。

1.10.2　div和span

HTML 有两个用来标出范围的标签：`<div>`和`<span>`。它们用来将内容划分成*语义性*（semantic）的块，也就是具有相似含义（meaning）的块。一个表格单元格或段落中的内容可能具有共同点，也可能没有，但是每个`<div>`和`<span>`中的内容应该具有相似的含义。

那么，`<div>`和`<span>`有什么区别呢？`<div>`是一个块级（block-level）元素，也就是说，它与前后元素之间有物理换行。但`<span>`不是块级的，它是*行内的*（inline），所以可以将它应用于句子中的一个短语。

1.10.3　class和id

在 HTML 文档中，将内容分隔为这些有意义的块。但是在此之后，仍然需要标识出那些需要修改其表现或行为的内容片段。为此，主要使用两个属性：`class` 和 `id`。CSS 和 JavaScript 都可以利用这些属性。CSS 样式表在规则中使用这些属性定义页面的外观，而 JavaScript 文件在代码中使用这些属性来影响页面上元素的行为。

- 类（class）标识出可能会多次使用的元素。例如，假设你要为电影院编写一个页面。可以为电影标题定义一个类，然后通过这个类指定标题应该是 14 像素、粗体和深蓝色的。

  ```
  .movieTitle {
    font: bold 14px;
    color: #000099;
  }
  ```

  应该将页面上的每个电影标题包围在一个标签中，并指定这个标题类型的 `class` 属性，如下所示：

  ```
  <p>We're currently showing <span class="movieTitle">The Aviator</span> and <span class="movieTitle">
  ➝The Outlaw</span>.</p>
  ```

- `id` 标识出的元素对于文档是唯一的。例如，如果在页面上电影院的名称只出现一次，那么可以使用 `id` 创建一个样式规则，定义电影院名称的外观是什么样的，如下所示：

```
#theaterName {
  font: bold 28px;
  color: #FF0000;
}
```

要显示电影院的名称时，只需在受影响的标签中添加这个 id 属性：

```
<h1 id="theaterName">The Raven Theater Presents:</h1>
```

上面示例中用于 CSS 的手段也可以应用于 JavaScript。在给 div 和 span（或任何其他元素）分配了 class 和 id 之后，就可以修改这些元素：不但可以用 CSS 修改它们的外观，还可以用 JavaScript 修改它们的行为。本书的其余部分主要讨论这个主题。

✔提示

- 有些人可能会在 CSS 中使用#和.时出现混淆，因为他们想不起哪个符号用于 class，哪个符号用于 id。我们的记忆方法是：在给定的页面上，一个 id 只能出现一次。“1”是一个数字，而井号（#）也称为数字符，所以这个符号用于 id。

1.11　要使用什么工具

因为 JavaScript 只是纯文本，所以可以使用几乎任何文本编辑器来编辑 JavaScript。甚至可以使用 Microsoft Word 这样的字处理程序，但是一定要确保 Word 将文件保存为 Text Only，而不是采用它自己的文件格式。HTML、JavaScript 和 CSS 文件必须是纯文本格式的，这样 Web 浏览器才能理解它们。

最好是使用以纯文本作为标准格式的程序。在 Windows 上，许多人使用记事本（见图 1-7）。在 Mac 上，可以使用 TextEdit，但是专业人员喜欢 Bare Bones 软件公司开发的 BBEdit（见图 1-8）。在 Unix 机器上，Emacs 是最好的文本编辑器之一。无论你使用什么程序，不要忘记用扩展名.html、.css 或.js 保存纯文本文件，这样才能顺利地将文件上传到 Web 服务器。

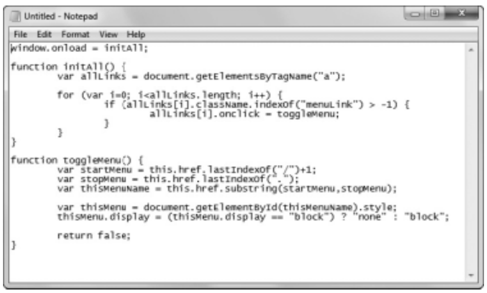


图 1-7　Windows 7 上的记事本



图 1-8　Mac OS X 上的 BBEdit

还可以使用某种 WYSIWYG（What You See Is What You Get，所见即所得）的 HTML 编辑器，比如 Adobe Dreamweaver。只需切换到 HTML Source 模式，就可以编写脚本了。

✔提示

- 如果你是 Mac 用户，那么试试 Bare Bones 软件公司（www.barebones.com）开发的 TextWrangler。它的特性不如 BBEdit 那么全面，但是它有一个大优点：它是免费的。
- 如果你想了解如何在 Dreamweaver 中使用代码工具的更多内容，推荐你阅读我们著的 *Dreamweaver CSS: Visual QuickStart Guide* 一书。

第 2 章

开　　始

2

热身够了，现在该编写脚本了。在本章中，你将学习把脚本放在 HTML 中的什么位置，如何在脚本中编写注释，让自己在过了一段时间之后仍然能够轻松地理解这些脚本，以及如何使用脚本与用户进行通信。还会看到如何使页面自动地转到另一个页面（这称为重定向，redirection）。我们开始吧！

目前需要了解的 HTML 基础知识见表 2-1。

表2-1　目前需要了解的HTML知识

标　　签	属　　性	意　　义
html		包含网页的HTML部分
head		包含网页的页头部分
script		包含网页的脚本或对外部脚本文件的引用。脚本常常是JavaScript
	src	外部脚本的位置
title		包含网页的标题
body		包含网页的内容
h1...h6		这些标签的内容作为标题信息。h1是最大尺寸的标题，h6是最小尺寸的标题
a		链接到另一个网页
	href	指定当单击链接时，用户应该转到哪里

2.1　将脚本放在哪里

脚本可以放在 HTML 页面上的两个位置：<head>和</head>标签之间（称为头脚本，header script），或者<body>和</body>标签之间（体脚本，body script）。脚本 2-1 是体脚本的一个示例。

脚本 2-1　脚本总是需要包围在<script>和</script> HTML 标签之间

```
<!DOCTYPE html>
<html>
<head>
     <title>My first script</title>
</head>
<body>
     <h1>
```

```
        <script>
            document.write("Hello, world!");
        </script>
    </h1>
</body>
</html>
```

有一个标出脚本的 HTML 容器标签，这个标签以<script>开头，以</script>结束。

⇛ 编写第一个脚本

1. <script>

这是 script 开始标签，告诉浏览器后面的代码是 JavaScript 而不是 HTML。

2. document.write("Hello, world!");

这是 JavaScript 的第一行：它获得文档窗口并在其中写入"Hello, world!"，如图 2-1 所示。请注意这一行末尾的分号（;），这告诉浏览器的 JavaScript 解释器这一行结束了。除了极少的例外情况，本书中每行 JavaScript 代码的末尾都使用分号。



图 2-1　“Hello, world!”示例是编程图书中的传统示例，我们也不想违反传统

3. </script>

这结束 JavaScript，并告诉浏览器后面的代码是 HTML。

✔**提示**

- script 标签的 language 属性和 type 属性（这里没有使用）已经废弃了，这意味着 W3C（负责的标准组织）已经将这个属性标记为在标准的未来版本中不必支持的属性，但还有不少旧脚本仍在使用它。
- 如果每一行上只有一个语句，那么在 JavaScript 行的末尾使用分号是可选的。为了使代码清晰，我们在本书中坚持使用分号（;）。由于同样的原因，我们建议你养成在代码中包含分号的习惯。
- 对于本书中的大多数地方，我们在代码解释中省略了<script>标签。可以看到，在脚本代码中仍然有这个标签，而且仍然需要它，但是我们不会反复解释它。
- 一个页面上可以有任意数量的<script>标签（因此有多个脚本）。

2.2　关于函数

在讲述下一个示例之前，你需要了解一下函数，在编写 JavaScript 时常常要用到它们。函数（function）是一组执行某一任务的 JavaScript 语句。每个函数必须有一个名称（除了一个非常少见的例外，这会在本书后面讨论），并可被脚本的其他部分调用。

在脚本运行期间，可以根据需要调用函数任意次。例如，假设你已经获得了用户在表单中输入的一些信息，并使用 JavaScript 将它保存起来了（在第 6 章中提供了关于这类问题的更多信息）。如果需要一次又一次地使用此信息，那么可以在脚本中反复使用相同的代码。但是，更好的方法是将这段代码编写成函数，然后在需要的时候调用这个函数。

函数由单词 function 加上函数名组成。函数名后面是圆括号，再后面是左花括号。组成函数内容的语句出现在后面的行上，然后用右花括号结束这个函数。函数的形式如下所示：

```
function saySomething() {
  alert("Four score and seven years ago");
}
```

注意到 alert 所在的行缩进了吗？这会使代码更易读。第一个花括号和最后一个花括号（你会注意到这两行没有缩进）之间的所有语句都是函数部分。这就是目前需要知道函数方面的内容。后续章节会介绍函数方面的更多内容。

2.3　使用外部脚本

在 HTML 页面上直接使用脚本（就像前面的示例那样）的问题是脚本只能供当前页面使用。因此，这种脚本有时候称为*内部脚本*（internal script）。但是，经常希望让多个 HTML 页面共享一个脚本。这要通过包含*外部脚本*（external script）的引用来实现，也就是只包含 JavaScript 的单独文件。这些外部文件称为.js 文件，因为无论它们叫什么，文件名都应该以.js 后缀结尾。各个页面只需在 script 标签中添加 src 属性，就可以调用.js 文件。

这就大大减少了每个页面上的代码，更重要的是，这会使站点更容易维护。当需要对脚本进行修改时，只需修改.js 文件，所有引用这个文件的 HTML 页面会自动响应修改。

在第一个外部脚本示例中，脚本 2-2 包含引用外部文件的 HTML，脚本 2-3 是外部 JavaScript 文件。

脚本 2-2　这段简单的 HTML 在 script 标签中包含对外部 JavaScript 文件的引用

```
<!DOCTYPE html>
<html>
<head>
     <title>My second script</title>
     <script src="script02.js"></script>
</head>
<body>
     <h1 id="helloMessage">
     </h1>
</body>
</html>
```

脚本 2-3　第一个外部 JavaScript 文件

```
window.onload = writeMessage;

function writeMessage() {
     document.getElementById("helloMessage").innerHTML = "Hello, world!";
}
```

⇛ 使用外部脚本

1. <script src="script02.js"></script>

这一行在脚本 2-2 中。在 script 标签中添加 src 属性会使浏览器寻找引用的文件。产生的网页看起来就像将脚本直接放在页面的 script 标签中一样，而实际上脚本放在外部的.js 文件中。

使用外部脚本只需要这一行。接下来，我们看看这个脚本中的内容。

2. `window.onload = writeMessage;`

在脚本 2-3 中，这一行的第一部分 `window.onload` 是一个事件处理程序，我们在第 1 章中讨论过了。等号后面是一个函数名 `writeMessage`。这一行的意思是“当窗口完成加载时，运行 `writeMessage` 函数”。

3. `function writeMessage() {`

这一行创建 `writeMessage()`函数。

4. `document.getElementById("helloMessage").innerHTML = "Hello, world!";`

再看看脚本 2-2，就会发现有一个`<h1>`标签，它的 `id` 为 `helloMessage`。在后面你会学到关于 `id` 的更多知识，目前只需知道 `id` 是它所属的元素在页面上的唯一标识符。换句话说，在给定的页面上，只能有一个元素具有这个特定的 `id`。这使 JavaScript 很容易通过使用 `getElementById()` 方法来获得和操作这个元素。`innerHTML` 属性仅仅是获得等号右边的字符串，并将它直接放到页面中，就像是我们在 HTML 中编写了这个字符串一样。所以，这行 JavaScript 代码的意思是“获得字符串`'Hello, world!'`，并将它放在文档中名为 `helloMessage` 的元素中”。结果如图 2-2 所示，跟图 2-1 一样。



图 2-2　将 JavaScript 转移到外部文件中的结果看起来与前一个示例一样，但这种方式更好

✔提示

- ❑ 支持外部 JavaScript 文件的浏览器包括：微软的 IE 4 及更高版本、Netscape 3 及更高版本，以及此后发布的几乎所有浏览器，包括 Firefox、Safari 和 Chrome 等现代浏览器。
- ❑ 有时候使用外部 JavaScript 文件这种方式来对用户隐藏 JavaScript。但是，如果用户的技术经验很丰富，有能力检查他们的浏览器缓存文件（浏览器已经看到的所有东西都存储在那里），那么这种方法是无效的。
- ❑ 在脚本 2-1（以及本书的先前版本）中，我们使用 `document.write()`将信息插入 HTML 页面。在后续版本中，我们主要使用设置 `innerHTML` 的方法，因为它更通用。一些人反对使用 `innerHTML` 属性，因为 W3C 已经废弃了它。但是，即使是这些心存疑虑的人也得承认，它是最简单的可以跨浏览器工作的方法，所以我们在本书中主要采用这种方法。第 10 章将介绍在 HTML 页面上添加或修改信息的“正式”方法。
- ❑ 如果你以前见过函数，那么可能会认为步骤 2 中的 `writeMessage` 引用应该是 `writeMessage()`。这种想法是不对的，因为这是两种不同的东西：带圆括号的函数名意味着正在调用这个函数；如果没有圆括号（就像这里的情况），就是将它赋值给事件处理程序，以便在此事件发生时运行它。

2.4　在脚本中添加注释

养成在脚本中添加注释的习惯是一种很好的做法。JavaScript 不会将插入的注释解释为脚本命令。尽管在编写脚本时，它可能看起来非常清晰，但是如果你过几个月再看这个脚本，就可能觉得它很晦涩。注释有助于解释你为什么按照某种方式解决问题。对脚本进行注释的另一个原因是，别人可能希望重用和修改你的脚本，注释对他们也有帮助。

脚本 2-4 给出了两种脚本注释的示例。第一种用于比较长的多行注释，第二种显示如何编写单行注释。

脚本 2-4　这里的代码演示如何给脚本添加注释，注释有助于开发者和其他开发人员理解代码

```
/*
    This is an example of a long JavaScript comment. Note the characters at the beginning and ending of the comment.

    This script adds the words "Hello,world!" into the body area of the HTML page.
*/

window.onload = writeMessage; // Do this when page finishes loading

function writeMessage() {
    // Here's where the actual work gets done

    document.getElementById("helloMessage").innerHTML = "Hello, world!";
}
```

注意，我们没有给出这个示例的 HTML，因为它与脚本 2-2 是相同的。从现在开始，如果 HTML 与前一个示例相比没有变化，我们就不再重复显示它了。

⇒ 对脚本进行注释

1. `/*`

```
This is an example of a long JavaScript comment. Note the characters at the beginning and
→ending of the comment.
 This script adds the words "Hello, world!" into the body area of the HTML page.
```

对于多行的注释，行开头的/*让 JavaScript 忽略此后的所有内容，直到注释的末尾为止。

2. `*/`

这是注释的末尾。

3. `window.onload = writeMessage;`

```
 // Do this when page finishes loading
 function writeMessage() {
    // Here's where the actual work gets done
    document.getElementById("helloMessage").innerHTML = "Hello, world!";
 }
```

这里是与前一个示例一样的脚本，但是加上了单行的注释。单行注释可以单独占据一行，也可以跟在代码行后面。在单行注释后面，同一行上不能再编写代码了；多行注释不能与代码同在一行上。

是的，我们也厌倦了“Hello, world!”示例，但这是传统，所有编程图书都以这个示例开始讲解。现在就来看点儿新鲜的东西吧！

2.5　向用户发出警告

JavaScript 的主要用途之一是向浏览站点的人提供反馈。可以创建一个弹出的警告窗口，向用户提供关于页面必须了解的重要信息。

在用户界面设计中，简单就是好。例如，你可以用一大片警告文本和巨大的动画引起用户的注意，但是这就有点儿物极必反了。相反，应该像脚本 2-5（HTML，它仅仅调用外部脚本）和脚本 2-6（JavaScript）这样，创建一个漂亮、简洁的警告窗口。现在，你知道我们为什么是技术作家而不是设计人员了。

脚本 2-5　这个示例的 HTML 包含<script>和<noscript>标签

```
<!DOCTYPE html>
<html>
<head>
     <title>My JavaScript page</title>
     <script src="script04.js"></script>
</head>
<body>
<noscript>
     <h2>This page requires JavaScript.</h2>
</noscript>
</body>
</html>
```

脚本 2-6　警告对话框帮助你与用户进行通信

```
alert("Welcome to my JavaScript page!");
```

⇛ 向用户发出警告

- ❑ alert("Welcome to my JavaScript page!");

是的，这就是所需的所有代码，其效果如图 2-3 所示。把希望显示的文本放在 alert()方法的直引号中就行了。

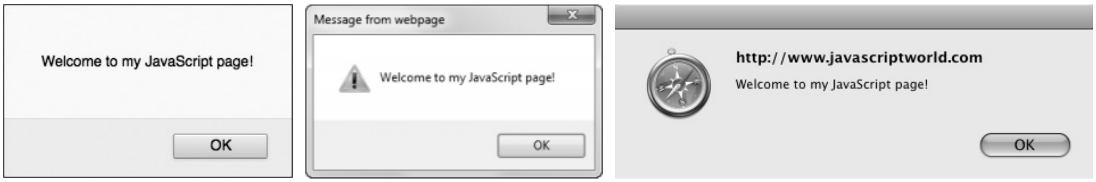


图 2-3　这个脚本只弹出一个对话框。显示的 3 个示例从上到下依次是在 Firefox 26、微软 IE 11 和 OS X 上的 Safari 6 中的对话框的外观

✓提示

- ❑ 在大多数 JavaScript 警告框中，会有某种迹象，让用户知道这个警告框是由 JavaScript 命令弹出的。这是一个安全特性，用来防止恶意的脚本编写者欺骗用户。无法用代码控制这个特性。例如，在 Safari 上，它显示打开这个警告的站点的 URL，如图 2-3 所示。Windows 和 Mac 上的 Chrome 采用同样的做法。在 IE 中，窗口标题总是“Message from webpage”。Firefox 则不显示信息，警告框直接覆盖在窗口顶部，并弹出警告信息。
- ❑ 这里还使用了<noscript>标签。在不支持 JavaScript 的浏览器（老式浏览器和关闭了 JavaScript 功能的浏览器）上，会显示一条消息，它指出这个页面需要 JavaScript。

2.6　确认用户的选择

向用户提供信息是有用的，但有时候还希望从用户那里获得信息。脚本 2-7 演示如何查明用户对你的问题的回答。这个脚本还引入了条件（conditional）的概念，在这里脚本进行一次测试，并根据测试的结果执行不同的操作。

脚本 2-7　可以根据用户对提示的反应给出相应的回复

```
if (confirm("Are you sure you want to do that?")) {
     alert("You said yes");
}
else {
     alert("You said no");
}
```

关于条件的更多信息

条件语句分成 3 部分：if 部分（在这里执行测试）、then 部分（在这里放置在结果为 true 时要执行的脚本命令）和可选的 else 部分（这里包含在测试结果不为 true 时要执行的脚本命令）。在 if 部分中，我们要测试的内容放在圆括号中，其他两部分的内容分别包含在花括号中。

⇛ 确认用户的选择

1. if (confirm("Are you sure you want to do that?")) {

confirm()方法有一个参数（向用户询问的问题），并根据用户的响应返回 true 或 false，如图 2-4 所示。

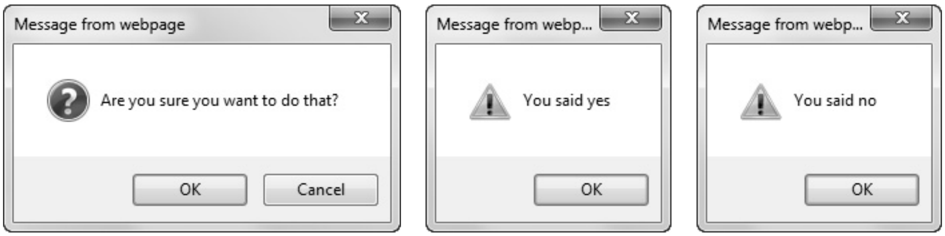


图 2-4　可以看到，你可以获得用户操作的结果，并在警告框中确认此结果。最上面的图向用户提出一个问题，单击 OK 或 Cancel 按钮的结果显示在下面

2. alert("You said yes");

如果用户单击 OK 按钮，那么 confirm()返回 true，因此出现的警告框中显示“You said yes”。可以看到，这是代码的 then 部分，尽管 JavaScript 中并没有 then 操作符。花括号就标出了 then 部分的范围。

3. }

这个花括号结束当 confirm()返回 true 值时执行的代码。

4. else {

这里开始仅当用户单击 Cancel 按钮时执行的代码部分。

5. alert("You said no");

如果用户单击 Cancel 按钮，那么 confirm()返回 false，并显示消息“You said no”。

6. }

这个花括号结束整个 if/else 条件语句。

✔提示

❑ 可以在 then 和 else 部分的花括号中放任意数量的语句。

不存在唯一正确的方式

编写任何脚本都有许多种方式，并且都能实现同样的效果。例如，当（且仅当）代码块中只有一条语句时，条件语句中就不需要使用花括号。

另外，还有一种可以替代条件语句的方法，其形式如下：

```
(condition) ? truePart : falsePart;
```

这相当于

```
if (condition) {
   truePart;
}
else {
   falsePart;
}
```

同样的简写方法也可以用来设置变量，例如：

```
myNewVariable = (condition) ? trueValue : falseValue;
```

这相当于

```
if (condition) {
   myNewVariable = trueValue;
}
else {
   myNewVariable = falseValue;
}
```

并不要求必须将花括号放在行尾或行首，true 和 false 代码块的缩进也不是必需的。这些都是样式问题，对于你最合适的样式就是正确的样式。

在本书中的大多数地方，为了保持代码清晰，我们在示例中包含花括号，而且选用比较长的条件语句形式。

2.7 提示用户

有时候，不是仅希望用户回答 Yes/No，而是希望得到更特定的响应。在这种情况下，可以像脚本 2-8 那样问一个问题（带默认回答），然后接收回复。图 2-5 显示其结果。

脚本 2-8 可以使用对话框询问用户并处理回复

```
var ans = prompt("Are you sure you want to do that?","");
if (ans) {
```

```
        alert("You said " + ans);
    }
    else {
        alert("You refused to answer");
    }
```



图 2-5　可以提示用户输入一个文本字符串，然后对这个字符串进行操作

⇒ 提示用户作出响应

1. `var ans = prompt("Are you sure you want to do that?","");`

这里声明了一个变量（在第 1 章中讨论过）。我们使用 var 关键字声明变量。在这个示例中，变量称为 ans，它被赋值为 prompt()的结果，也就是用户在提示对话框中输入的文本。

传递给 prompt()方法的是由逗号分隔的两段信息（正式的术语是参数）：向用户询问的问题和默认回答。这个方法返回用户的响应或 null。当用户单击 Cancel 按钮，或当没有默认回答而用户单击了 OK 按钮，再或当用户清除了默认回答并单击了 OK 按钮时，就会出现 null。有些浏览器会在提示对话框上显示一个关闭控件，使用这个控件也会返回 null 结果。

2.
```
if (ans) {
    alert("You said " + ans);
}
```

这个条件语句使用了刚才设置的变量。如果 ans 存在（也就是说，用户输入了响应），那么会弹出一个警告对话框，其中显示"You said "（请注意这个文本字符串末尾的额外空格）和 ans 值的拼接。

3.
```
else {
    alert("You refused to answer");
}
```

如果 ans 是 null（因为用户没有在提示对话框中输入任何内容，或者单击了 Cancel 按钮），那么执行条件语句的 else 块，并弹出警告对话框。

✔提示

- 使用 var 有如下两种作用。
  - 它让 JavaScript 创建一个变量（也就是在内存中为这个新对象留出一些空间）。
  - 它定义变量的*作用域*（scope），也就是 JavaScript 在哪些地方需要知道这个对象的内容（见下面的补充内容“作用域是什么”）。如果变量是在一个函数中创建的，那么它是这个函数的*局部*（local）变量，其他函数不能访问它。如果它是在任何函数之外创建的，它就是*全局的*（global），脚本中的所有代码都可以访问它。在这个脚本中，我们创建了 ans 全局变量。

- ❑ 在某些浏览器中，如果省略 prompt 的第二个参数（默认响应），那么一切正常。但是在其他浏览器中，出现的提示窗口中会显示默认值“undefined”。解决方案是总是包含某个默认值，即使是空字符串也可以（就像脚本 2-8 中一样）。
- ❑ Chrome 和 Firefox 允许访问者禁止页面创建其他对话框（参见图 2-5）。因此，如果你创建的页面依赖用户查看对话框，那可不好。

作用域是什么

在世界上的大多数地方，如果提到 Broadway（大道，或特指百老汇），人们都知道你指的是纽约的一条大街。尽管这条大街本身在纽约，但是全世界的人都知道你的意思。可以认为 Broadway 这个词是全局的。

但是，如果你在加利福尼亚的圣迭戈，当你提及 Broadway 时，人们会认为 Broadway 指的是当地市区的一条主要大街。这是一个局部值。在圣迭戈，你指的是局部意义的 Broadway，还是全局意义的 Broadway，这一点并不明确，因此可能导致混淆。

如果你在圣迭戈，那么默认的是局部意义。在指全局意义时，必须明确地说“纽约的 Broadway”。在圣迭戈之外的地方，人们首先会想到纽约的百老汇，除非他们本地也有名为 Broadway 的大街。

一条大街的作用域（也就是通常所说的范围）是一个地区，在这个地区中这条大街是默认意义；也就是说，如果没有加上其他修饰词，那么提到这条大街的名字，人们就会自动地想到这条大街。圣迭戈的 Broadway 的作用域是局部的，即圣迭戈市内和附近的郊区。纽约的 Broadway 的作用域是全局的，即世界上任何地方的人都知道你指的是什么地方。

在 JavaScript 代码中，要想避免与变量作用域有关的问题和混淆，最容易的方法是避免使用同名的两个变量在不同的地方做不同的事。如果必须使用同名的变量，就一定要弄清变量的作用域！

2.8　用链接对用户进行重定向

可以根据用户是否打开了 JavaScript 功能，无缝地对用户进行重定向（redirection），也就是将用户转到另一个页面。这个示例演示如何将重定向功能嵌入链接中。我们将使用两个 HTML 页面和一个 JavaScript 文件。第一个 HTML 页面（见脚本 2-9）向用户显示链接。脚本 2-10 是 JavaScript 文件，脚本 2-11 是在用户启用了 JavaScript 功能的情况下用户被重定向到的 HTML 页面。当用户单击这个链接（见图 2-6）时，根据他们是否打开了 JavaScript 功能，将被带到两个页面之一。



图 2-6　这个页面上的链接包含重定向代码

脚本 2-9　这个 HTML 页面基于链接对用户进行重定向

```
<!DOCTYPE html>
<html>
<head>
```

```
    <title>Welcome to our site</title>
    <script src="script07.js"></script>
</head>
<body>
    <h2>
       <a href="script04.html" id="redirect"> Welcome to our site... c'mon in!</a>
    </h2>
</body>
</html>
```

脚本 2-10　通过将重定向功能嵌入在代码中，用户甚至不知道你的脚本干预了链接的行为

```
window.onload = initAll;

function initAll() {
    document.getElementById("redirect").onclick = initRedirect;
}


function initRedirect() {
    window.location = "jswelcome.html";
    return false;
}
```

脚本 2-11　这是启用了 JavaScript 功能的用户将看到的 HTML 页面

```
<!DOCTYPE html>
<html>
<head>
    <title>Our site</title>
</head>
<body>
    <h1>Welcome to our web site, which features lots of cutting-edge JavaScript</h1>
</body>
</html>
```

⇒ 对用户进行重定向

1. `<a href="script04.html" id="redirect">Welcome to our site... c'mon in!</a>`

在脚本 2-9 中，这是用户单击的链接。如果用户没有启用 JavaScript 功能并单击链接，那么他们会按照通常的 href 路径前进，到达如图 2-7 所示的页面。如果用户启用了 JavaScript 功能并单击链接，那么脚本（见步骤 4）就会发挥作用并加载一个新页面。



图 2-7　如果你认为 JavaScript 是站点的必要部分，就用这个消息向用户说明情况

2. `window.onload = initAll;`

这行代码在脚本 2-10 中。当完成页面加载时，它会触发 `initAll()`函数。

3.
```
function initAll() {
   document.getElementById("redirect").onclick = initRedirect;
}
```

这个函数告诉 `id` 为 `redirect` 的元素（也就是步骤 1 中的链接），在它被单击时应该调用 `initRedirect()` 函数。

4.
```
function initRedirect() {
    window.location = "jswelcome.html";
    return false;
}
```

如果调用这个函数，它就将 `window.location`（即浏览器中显示的页面）设置为一个新页面。`return false` 表示停止对用户单击的处理，这样就不会加载 `href` 指向的页面。

这种方式最酷的特色是，我们完成了重定向而用户根本不会意识到发生了重定向。他们仅仅是根据自己的情况到达了两个页面之一。如果他们启用了 JavaScript 功能，就会到达图 2-8 所示的页面。



图 2-8 支持 JavaScript 功能的浏览器会显示这个页面

✔提示

- 初看上去，似乎可以只在全局范围设置 `onclick` 处理程序，即在加载页面时，但是不能这么做。浏览器有可能还没有遇到 `id` 为 `redirect` 的元素，尤其是对于复杂的大页面，在这种情况下，JavaScript 就不能设置 `onclick` 处理程序。所以，我们必须等待页面完成加载，这要通过 `onload` 实现。
- 请记住，一些用户习惯在鼠标位于链接上时看到链接指向的页面，不喜欢被自动转到其他页面。

2.9 使用 JavaScript 改进链接

有时候，在用户单击链接之后，但在浏览器转到链接的目的地之前，希望执行某种操作。典型的示例是，在用户进入站点上的特定页面之前发出警告，或者在用户离开站点时给出明确的提示。在这个示例中，我们将弹出一个警告对话框，然后再转到最终的目的地。脚本 2-12 显示 HTML，脚本 2-13 显示需要对前面的脚本做的少量修改。

脚本 2-12　与平常一样，HTML 在链接标签中包含 JavaScript 可以使用的 id

```
<!DOCTYPE html>
<html>
<head>
     <title>Welcome to our site</title>
     <script src="script08.js"></script>
</head>
<body>
     <h2>
        Hey, check out <a href="http://www.pixel.mu" id="redirect"> my cat's Web site</a>.
     </h2>
</body>
</html>
```

脚本 2-13　链接改进脚本

```
window.onload = initAll;

function initAll() {
     document.getElementById("redirect").onclick = initRedirect;
}

function initRedirect() {
     alert("We are not responsible for the content of pages outside our site");
     window.location = this;
     return false;
}
```

⇛ 改进链接

1. Hey, check out my cat's Web site.

脚本 2-12 中的这行代码显示链接，其中包括链接目的地的 href 和链接的 id，脚本 2-13 将使用这个 id。页面见图 2-9。

2. alert("We are not responsible for the content of pages outside our site");

在单击链接之后，会显示这个警告，如图 2-10 所示。



图 2-9　单击这个链接就会把用户重定向到猫的 Web 站点

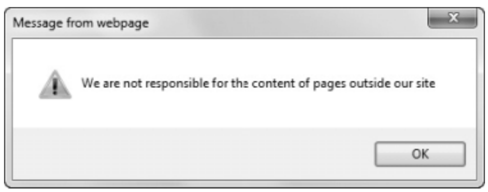


图 2-10　如果用户使用支持 JavaScript 的浏览器，那么会在离开站点时看到这个警告消息

3. `window.location = this;`

这一行将浏览器窗口设置为关键字 this 指定的位置，this 包含这个链接。目前，只需将 this 看作一个容器，本书后面会详细介绍它。如果你现在就想知道更多的信息，那么可以阅读后面的补充内容“this 是什么”。当用户到达最终目的地时，页面像图 2-11 这样（至少，使用猫的网页作为目的地）。



图 2-11　在这个示例中，这只猫的页面实际上是由我们维护的（你不会认为它自己会编写页面吧）

this 是什么

在这个示例中使用了 this，但是 this 究竟是什么还不很清楚。

JavaScript 关键字 this 使脚本能够根据使用这个关键字的上下文将值传递给函数。在这个示例中，this 是在一个由标签的事件触发的函数中使用的，所以 this 是一个链接对象。在后面的示例中，将看到在其他地方使用 this，你应该能够根据使用它的上下文判断出 this 是什么。

✔提示

- ❑ 你可能会注意到，代码中并没有引用特定的网页——这是 this 关键字的作用之一。this 替我们完成的工作之一是从 HTML 链接获得 URL（也就是 a 标签的 href 属性值）。由于采用了这种方式，如果以后将脚本 2-12 改为指向其他页面而不是猫页面，就不必修改脚本 2-13。实际上，可以让 Web 站点上的所有链接都调用这个相同的脚本，这一行代码都会自动获得相应的 href 值。
- ❑ 如果前面的提示解释得还不够，可以这样考虑：按照这种方式，可以使用 WYSIWYG 编辑器修改你的 HTML 页面，而且修改者可以完全不了解 JavaScript。只要他们只是修改 HTML 页面，就不会破坏你的脚本。
- ❑ 这对你来说还不够吗？还有一个好处：如果用户的浏览器不理解 JavaScript，那么它只会加载 HTML 页面，而不显示警告。当他们单击链接时，会像一般情况下那样加载页面：不会发生错误，也不会提示“你必须换用其他浏览器”，没有任何问题。
- ❑ 这种编程方式称为*无干扰脚本编程*（unobtrusive scripting），它将代码与 HTML 分隔开，从而使这两者都更加灵活。如果想了解关于 Web 编程的更多术语，请阅读后面的补充内容“常见术语”。

常见术语

使用 JavaScript 一段时间后，你可能被不断增加的术语弄得头昏，其中许多术语在根本上是关于 JavaScript 是什么以及不是什么的。下面快速总结一下术语问题，帮助你澄清概念。（如果你不觉得这是个迫切的问题，那么和一些脚本开发人员聊一会儿吧！）

- JavaScript。尽管正式地说这个术语是属于 AOL 的（AOL 收购 Netscape），但是它常常用作所有与 JavaScript 类似的脚本技术，比如微软的 JScript。我们在本书中也沿用这种做法[①]。
- DHTML。即 Dynamic HTML（动态 HTML），但是在现实中，它的实际意义取决于说话的人。Web 标准项目将 DHTML 定义为："……一种过时的脚本技术，它的主要特征是它会改变某些元素的样式属性，以及使用浏览器特定的 DOM document.layers 和 document.all。"幸好这个术语我们已不再使用了。
- DOM 脚本编程（DOM scripting）。一种使用 JavaScript 编写网页的方式，按照这种方式，代码只通过操纵 W3C DOM 来修改页面（也就是说，不使用专有的、非标准或已废弃的属性）。脚本 2-10 和脚本 2-13 引用 `document.getElementById("redirect").onclick`，这就是 DOM 脚本编程的做法。
- 无干扰脚本编程（unobtrusive scripting）。一种使用 JavaScript 编写网页的方式，按照这种方式，网页的行为和它的内容是分隔开的——也就是说，HTML 在一个文件中，JavaScript 在另一个文件中。这是一种最佳实践，它的作用就像是对 HTML 和 CSS 进行分隔，即页面的表现方式（CSS）在一个文件中，内容（HTML）在另一个文件中。
- 渐进增强。"无干扰编程"的一种附加功能。如果代码编写好之后，没有 JavaScript 支持（或使用功能比较差的浏览器）的访问者也能够使用站点的所有功能，只是用户体验稍差，这就叫渐进增强。脚本 2-10 和脚本 2-13 也是渐进增强的例子。在这些示例中，没有 JavaScript 支持的访问者也可以单击链接，但是如果有 JavaScript 支持，用户体验会更丰富。

在本书中，我们要使用许多种脚本技术。尽管我们建议采用无干扰脚本编程/渐进增强方式（而且尽可能演示这种做法），但是我们也知道你（作为刚入门的脚本开发人员）主要需要理解和支持比较老式但容易编写的代码。在真实环境中，我们也知道最直接的方式往往就是最简单的方式。所以我们在脚本 2-3 中使用了 `innerHTML`，尽管它不属于 W3C DOM。

2.10　使用多级条件

有时候，在一个条件测试中需要两个以上的选择，仅 `then` 和 `else` 是不够的。尽管可以嵌套 `if` 语句，但更简单的方法常常是使用 `switch/case` 语句。`switch/case` 构造允许针对多个值检查一个变量。如图 2-12 所示，这个脚本根据用户单击的按钮，在警告对话框中返回 3 段不同的总统语录之一。脚本 2-14 是 HTML，这相当简单。脚本 2-15 是 JavaScript，它使用 `switch/case` 构造区分不同的总统语录。

① 从技术上说，JavaScript 是 ECMAScript 语言标准的实现。Adobe 的 ActionScript 是该标准的另一种实现。——编者注

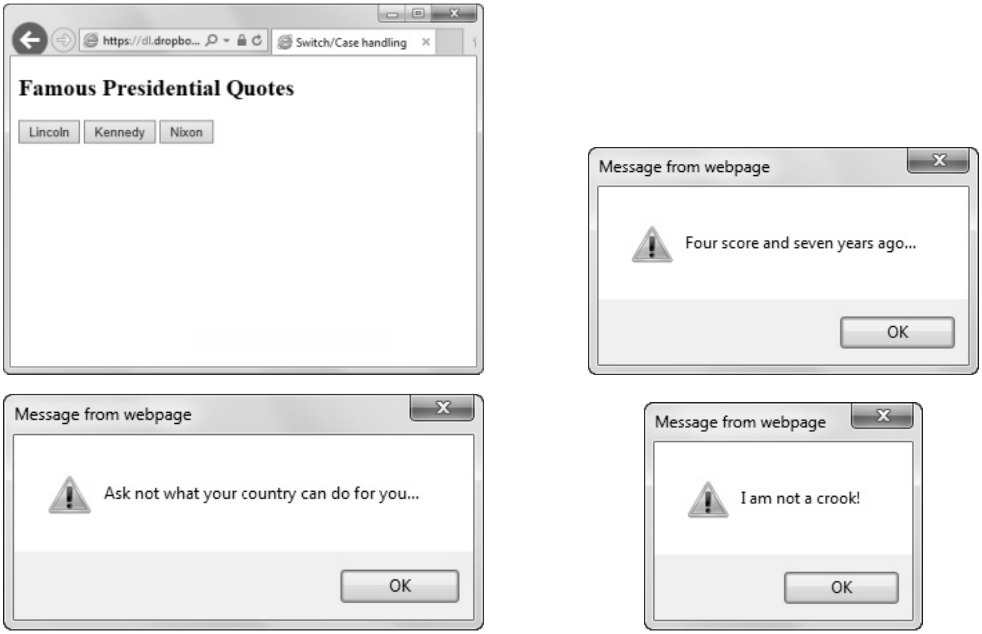


图 2-12　在最上面的窗口中，调用 3 个按钮的函数会产生下面 3 个对话框所示的 3 种不同响应

脚本 2-14　这段 HTML 建立多级条件的页面

```
<!DOCTYPE html>
<html>
<head>
     <title>Switch/Case handling</title>
     <script src="script09.js"></script>
</head>
<body>
<h2>Famous Presidential Quotes</h2>
<form action="#">
     <input type="button" id="Lincoln" value="Lincoln">
     <input type="button" id="Kennedy" value="Kennedy">
     <input type="button" id="Nixon" value="Nixon">
</form>
</body>
</html>
```

脚本 2-15　这种条件语句允许针对多种可能性进行检查

```
window.onload = initAll;

function initAll() {
     document.getElementById("Lincoln").onclick = saySomething;
     document.getElementById("Kennedy").onclick = saySomething;
     document.getElementById("Nixon").onclick = saySomething;
}

function saySomething() {
```

```
        switch(this.id) {
            case "Lincoln":
                alert("Four score and seven years ago...");
                break;
            case "Kennedy":
                alert("Ask not what your country can do for you...");
                break;
            case "Nixon":
                alert("I am not a crook!");
                break;
            default:
        }
    }
```

⇒ 使用 switch/case 语句

1. `window.onload = initAll;`

当加载页面时，调用 `initAll()`函数。

2.
```
function initAll() {
    document.getElementById("Lincoln").onclick = saySomething;
    document.getElementById("Kennedy").onclick = saySomething;
    document.getElementById("Nixon").onclick = saySomething;
```

在这个函数中，我们为页面上的每个按钮设置了 onclick 处理程序。因为在 HTML 中设置了 id 属性和 value 属性，所以可以使用 getElementById()设置事件处理程序。如果有 value 属性，就可以使用 getElementByValue()调用，那么就不必设置 id 属性。

3. `function saySomething() {`

这一行开始 saySomething()函数。

4. `switch(this.id) {`

this 对象的 id 用作 switch()的参数。这个值将决定执行以下 case 语句中的哪一个。

5.
```
case "Lincoln":
    alert("Four score and seven years ago...");
    break;
```

如果 this 对象的 id 是 Lincoln，那么显示这个警告消息。如果用户单击 Lincoln，就会进入这里的代码。但是，在这里已经执行了我们需要的操作，所以我们希望离开这个 switch 语句，为此，需要使用 break 语句。如果没有 break，就会继续执行下面的所有代码。尽管在某些情况下继续执行下面的分支是我们需要的效果，但是在这个示例中不应该这么做。

6.
```
case "Kennedy":
    alert("Ask not what your country can do for you...");
    break;
```

如果用户单击 Kennedy，就会进入这个 case 块。

7.
```
case "Nixon":
    alert("I am not a crook!");
    break;
```

最后，如果用户单击 Nixon，就会进入这里，这里弹出另一个警告对话框，然后退出 switch 语句。

8. `default:`

如果用户的输入与上面的条件都不匹配，那么就会执行这里的代码。也就是说，如果 switch 值与任何 case 值都不匹配，就会进入 default 部分。default 块是可选的，但是包含 default 块是一种好的编程习惯，可以以防万一。在这个脚本中，这里没有要执行的代码，因为我们应该不可能进入 default 部分。

9. `}`

这个右花括号结束 switch 语句。

✔提示

- 也可以向 switch 语句传递字符串之外的其他值。可以在 switch 语句中使用数字值，甚至对数学计算的结果进行评估。但是，如果结果应该是数字，那么要确保 case 语句是匹配的——case 语句应该检查数字，而不是字符串（例如 5，而不是"5"）。

2.11 处理错误

即使你有多年使用计算机的经验，但是你站点的许多用户很可能没什么经验。因此，应该向他们提供有意义的错误消息，而不是大多数浏览器在拒绝用户的操作时返回的莫名其妙的消息。脚本 2-16 演示如何使用 JavaScript 的 try/throw/catch 命令产生友好、有用的错误消息。我们将这个功能放在一个简单的平方根计算器中。

脚本 2-16 这个脚本用 JavaScript 适当地处理错误

```
window.onload = initAll;

function initAll() {
     var ans = prompt("Enter a number","");
     try {
        if (!ans || isNaN(ans) || ans<0) {
           throw new Error("Not a valid number");
        }
        alert("The square root of " + ans + " is " + Math.sqrt(ans));
     }
     catch (errMsg) {
        alert(errMsg.message);
     }
}
```

⇛ 适当地处理错误

1. `var ans = prompt("Enter a number", "");`

这是一个普通的提示，我们把它存储在 ans 变量中供以后使用。在这个示例中，我们希望用户输入一个数字。如果用户确实输入了合适的数字，JavaScript 就会显示所输入数字的平方根。

2. `try {`

但是，如果他们没有输入数字（如图 2-13 所示），那么代码能够捕捉到这一错误并显示有意义的消息。即使用户在要求他们输入数字时输入了单词，我们也能够有礼貌地发出提示。首先使用 try 命令。在这块代码中，我们检查用户的输入是否有效。



图 2-13　我们要求用户输入数字，但是用户可能输入任何内容，比如非数字项

3.
```
if (!ans || isNaN(ans) || ans<0) {
    throw new Error("Not a valid number");
  }
```

我们要关注 3 种情况：根本没有输入；用户输入了某些内容，但不是数字；输入的是数字，但它是负数（因为负数的平方根是虚数，这超出了这个示例的范围）。如果`!ans`是`true`，就意味着用户没有输入任何内容。内置的`isNaN()`方法检查传递给它的参数是否“不是数字”（Not a Number）。如果`isNaN()`返回`true`，就说明输入的内容是无效的。如果`ans`小于`0`，它就是负数。对于以上任何情况，都希望抛出一个错误，指出“Not a valid number”（不是有效的数字）。当抛出错误之后，JavaScript 跳出`try`块并寻找对应的`catch`语句。因此，`try`块中其余的代码都被跳过。

4.
```
alert("The square root of " + ans + " is " + Math.sqrt(ans));
```

如果输入了有效的内容，就显示平方根，如图 2-14 所示。

5.
```
}
```

这个右花括号结束`try`块。

6.
```
catch (errMsg) {
    alert(errMsg.message);
  }
```

这是出现错误时要使用的`catch`语句。`error`作为参数传递给它，它显示错误的`message`部分（见图 2-15）。如果没有抛出错误，`catch`中的代码就不会执行。

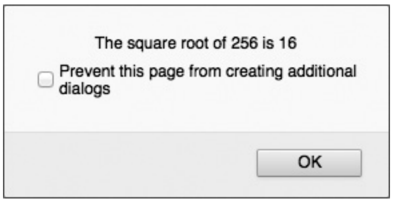


图 2-14　这是输入有效数字时脚本的结果

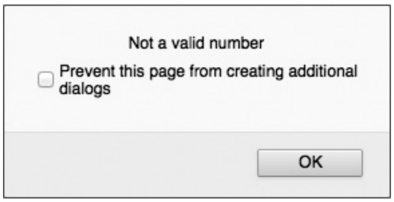


图 2-15　如果输入了错误的数据，就告诉用户

✓提示

- 还有一个可选部分：最后的{}块。这个部分放在`catch`后面，无论`try`块是否抛出错误，这里包含的代码都应该执行。
- 下一章会详细介绍`Math`对象。

第 3 章

第一个 Web 应用程序

对JavaScript 的作用有所体会之后，我们来更深入地探索 JavaScript 语言。在本章中，我们将详细讨论 JavaScript 的基本元素并介绍 JavaScript 语言的其他方面，比如循环、数组和函数（不必担心，这里的内容并不难理解）。

你还会学习如何使用 JavaScript 编写网页，JavaScript 如何处理用户所犯的错误，等等。

目前需要了解的 HTML 知识见表 3-1。

表3-1　目前需要了解的HTML知识

标　　签	意　　义
`table`	在网页上显示表格数据
`tr`	在表格中开始一行
`th`	表格中列的标题单元格
`td`	包含表格中的每个单元格

3.1　用循环进行重复操作

在编程中，常常需要测试某一条件，并且根据需要多次重复执行测试。我们用一个例子来说明：在字处理程序中进行查找和替换。查找一段文本，将它改为另一个文本字符串，并对文档中出现第一个字符串的所有地方重复这个过程。现在，假设你用一个程序自动完成这个任务。这个程序要执行一个循环（loop），从而以指定的次数重复一个操作。在 JavaScript 中，循环是基本的编程特性。

关于循环的更多知识

在本书中，我们最常用的循环类型是 `for` 循环，这种循环以 `for` 命令开始。这种循环使用一个计数器（counter），计数器是一个变量，它最初是某个值（常常是零）。这种循环在测试条件得不到满足时就会结束。

在循环结构开头的命令后面是圆括号。在圆括号中常常有计数器定义和递增计数器的方式（也就是增加计数器值的方式）。

在后面几个示例中，我们将构建一个简单而且大家熟悉的应用程序：Bingo 卡片游戏。我们使用每个示例演示 JavaScript 的不同方面。先来看一个 HTML 页面，如脚本 3-1 所示。它包含的表格就是

Bingo 卡片的框架（见图 3-1）。看一下这个脚本，你会发现第一行包含卡片顶部的字母，后续的每一行包含 5 个表格单元格。大多数单元格只包含一个非中断空格（使用 HTML 实体` `），但是第三行包含一个 Free 格，所以这一行中的一个表格单元格包含单词 Free。注意，每个单元格都有 `id` 属性，脚本使用这个属性操纵单元格的内容。`id` 采用的形式是 `square0`、`square1`、`square2`，直到 `square23`，采用这种形式的原因在后面解释。在页面的底部，有一个用来生成新卡片的链接。



图 3-1　这个 Bingo 卡片已经随机生成了许多数字，但它还不是有效的 Bingo 卡片

脚本 3-1　这个 HTML 页面建立 Bingo 卡片的框架

```
<!DOCTYPE html>
<html>
<head>
     <title>Make Your Own Bingo Card</title>
     <link rel="stylesheet" href="script01.css">
     <script src="script01.js"></script>
</head>
<body>
<h1>Create A Bingo Card</h1>
<table>
     <tr>
        <th>B</th>
        <th>I</th>
        <th>N</th>
        <th>G</th>
        <th>O</th>
     </tr>
     <tr>
        <td id="square0"> </td>
        <td id="square5"> </td>
        <td id="square10"> </td>
        <td id="square14"> </td>
        <td id="square19"> </td>
     </tr>
     <tr>
        <td id="square1"> </td>
        <td id="square6"> </td>
        <td id="square11"> </td>
```

```
            <td id="square15"> </td>
            <td id="square20"> </td>
        </tr>
        <tr>
            <td id="square2"> </td>
            <td id="square7"> </td>
            <td id="free">Free</td>
            <td id="square16"> </td>
            <td id="square21"> </td>
        </tr>
        <tr>
            <td id="square3"> </td>
            <td id="square8"> </td>
            <td id="square12"> </td>
            <td id="square17"> </td>
            <td id="square22"> </td>
        </tr>
        <tr>
            <td id="square4"> </td>
            <td id="square9"> </td>
            <td id="square13"> </td>
            <td id="square18"> </td>
            <td id="square23"> </td>
        </tr>
    </table>
    <p><a href="script01.html" id="reload">Click here</a> to create a new card</p>
    </body>
    </html>
```

脚本 3-2 是我们用来对 Bingo 卡片的内容应用样式的 CSS 文件。如果你不了解 CSS，也不需要担心，因为在这里 CSS 并不重要。对于其他 Bingo 卡片示例，HTML 和 CSS 页面没有变化，所以我们只在这里显示它们一次。

脚本 3-2 这个 CSS 文件为 Bingo 卡片添加样式

```
body {
     background-color: white;
     color: black;
     font-size: 20px;
     font-family: "Lucida Grande", Verdana, Arial, Helvetica, sans-serif;
}

h1, th {
     font-family: Georgia, "Times New Roman", Times, serif;
}

h1 {
     font-size: 28px;
}

table {
     border-collapse: collapse;
}

th, td {
     padding: 10px;
     border: 2px #666 solid;
     text-align: center;
     width: 20%;
```

```
}

#free, .pickedBG {
     background-color: #f66;
}

.winningBG {
     background-image:url(images/redFlash.gif);
}
```

这个示例演示如何建立和使用循环，用随机生成的数字填充 Bingo 卡片的内容。脚本 3-3 包含所需的 JavaScript。这个脚本生成的卡片不是有效的 Bingo 卡片，因为 Bingo 卡片游戏对特定列中的数字有一些限制。后面的示例将对脚本进行改进，直到产生有效的 Bingo 卡片。

脚本 3-3　欢迎你来到第一个 JavaScript 循环

```
window.onload = initAll;

function initAll() {
     for (var i=0; i<24; i++) {
        var newNum = Math.floor(Math.random() * 75) + 1;

        document.getElementById("square" + i).innerHTML = newNum;
     }
}
```

Bingo 卡片中的内容

你肯定见过 Bingo 卡片，但是可能没有仔细研究过。美国的 Bingo 卡片是 5×5 的方形，5 个列上标着 B-I-N-G-O，格子中包含 1～75 的数字。正中间通常是一个空的格子，常常印着单词 free。每列可以包含的数字的范围如下：

- B 列包含数字 1～15；
- I 列包含数字 16～30；
- N 列包含数字 31～45；
- G 列包含数字 46～60；
- O 列包含数字 61～75。

⇒ 使用循环创建表格内容

1. `window.onload = initAll;`

这行代码在脚本 3-3 中。当窗口完成加载时，它调用 `initAll()`函数。使用事件处理程序调用函数是常用的做法。

2. `function initAll() {`

这一行开始函数。

3. `for (var i=0; i<24; i++) {`

这一行开始循环。程序员通常使用变量 `i` 作为循环内部的计数器变量。首先，将 `i` 设置为 0。分号表示这个语句结束了，这使我们能够在同一行上放置另一个语句。下一部分的意思是，“如果 `i` 小于 24，那么执行花括号中的代码”。第二个分号后面的部分将 `i` 的值加 1。因为这是第一次用到递增，

我们来解释一下。i++部分使用第 1 章介绍过的++操作符将 i 的值加 1。这个循环将重复 24 次，所以循环中的代码将执行 24 次。在第一次迭代中，i 是 0；在最后一次迭代中，i 是 23。

4. `var newNum = Math.floor(Math.random() * 75) + 1;`

在循环内部，我们创建一个新的变量 newNum，并将它赋值为等号右边计算的结果。内置的 JavaScript 命令 Math.random()生成 0 ~ 1 的一个随机数，比如 0.123456789。将 Math.random()与最大值相乘（请记住，Bingo 卡片中的值是从 1 到 75）会生成 0 到最大值之间的结果。对结果进行 floor 运算会获得结果的整数部分，即 0 到（最大值减 1）的整数。再加 1 就会获得 1 到最大值的数字。

5. `document.getElementById("square" + i).innerHTML = newNum;`

在这里，我们将刚才获得的随机数写入表格中。我们处理的元素的 id 属性是 square 再拼接上 i 的当前值。例如，在循环的第一次迭代中，i 的值是 0，所以这一行处理 id 为 square0 的元素。这一行将 square0 对象的 innerHTML 属性设置为 newNum 的当前值。然后，因为我们还在循环中，步骤 4 和步骤 5 会重复执行，直到整个 Bingo 卡片填写完毕。

循环的组成

for 循环有 3 个组成部分，如图 3-2 所示。



图 3-2　循环的 3 个组成部分

1. **初始化步骤**。在循环的第一次迭代中，这个部分对循环变量（在这个示例中是 i）进行设置。

2. **限制步骤**。在这里指出什么时候停止循环。一般人是从 1 数到 10，但是在编程语言中常常是从 0 数到 9。在这两种情况下，循环中的代码都会运行 10 次，但是对于数组从 0 位置开始的语言（比如 JavaScript），后一种方法更合适。这就是循环的限制条件为“小于 userNum”而不是“小于等于 userNum”的原因。假设变量 userNum 是 10，并且希望循环运行 10 次。如果从 0 数到 9（使用“小于 userNum”测试），那么循环运行 10 次。如果从 0 数到 10（使用“小于等于 userNum”测试），那么循环运行 11 次。

3. **递增步骤**。在这里指定在循环的每次迭代中循环变量增加多少。在这个示例中，使用++每次将 i 的值加 1。

表 3-2 列出了常用的 JavaScript Math 函数。

表3-2　JavaScript Math函数

函　　数	描　　述
abs	绝对值
sin、cos、tan	标准三角函数，参数用弧度表示
acos、asin、atan	反三角函数，返回值用弧度表示
exp、log	以e为底数的指数和自然对数
ceil	返回大于等于当前参数的最小整数

（续）

函　　数	描　　述
floor	返回小于等于当前参数的最大整数
min	返回两个参数中较小者
max	返回两个参数中较大者
pow	指数函数，第一个参数是底数，第二个参数是幂
random	返回介于0和1之间的随机数
round	返回当前参数最接近的整数，四舍五入
sqrt	平方根

3.2　将值传递给函数

常常需要获得一些信息并将它交给函数使用，这称为将信息传递（pass）给函数。例如，看一下这个函数定义：

```
function playBall(batterup)
```

变量 batterup 是函数的参数。当调用函数时，可以将值传递给函数。当处于函数内部时，此数据就保存在 batterup 变量中。可以向函数传递需要使用的任何数据，包括文本字符串、数字，甚至其他 JavaScript 对象。例如，可以通过 batterup 变量传递文本字符串形式的玩家名称（"Mantle"），或者他在玩家中的编号（7）（但是将这两种形式混合使用是非常糟糕的想法，除非你确实知道自己在做什么）。与所有变量一样，应该给用作函数参数的变量起合适的名字，这个名称应该能够说明变量的用途。

在一个函数中可以有多个参数，只需将它们放在圆括号中并用逗号分隔：

```
function currentScore(hometeam,visitors)
```

下面这 3 个代码片段是等效的：

```
currentScore(6,4);

var homeScore = 6;
var visitingScore = 4;
currentScore(homeScore,visitingScore);

currentScore(6,3+1);
```

对于这 3 个示例，在 currentScore()中，hometeam 的值都是 6，visitors 的值都是 4（这对于主队来说是好消息）。

在这个示例中，我们将对脚本 3-3 作一些调整，从 initAll()函数中去掉一些计算，将它们转移到一个带参数的函数中，这样代码的作用就更清晰了。产生的脚本见脚本 3-4。

脚本 3-4　通过将值传递给 setSquare()函数，脚本更容易阅读和理解了

```
window.onload = initAll;

function initAll() {
     for (var i=0; i<24; i++) {
        setSquare(i);
     }
}
```

```
function setSquare(thisSquare) {
     var currSquare = "square" + thisSquare;
     var newNum = Math.floor(Math.random() * 75) + 1;

     document.getElementById(currSquare). innerHTML = newNum;
}
```

⇛ 将值传递给函数

1. setSquare(i);

这行代码在 initAll()函数中。我们将 i 的值传递给 setSquare()函数。

2. function setSquare(thisSquare) {

这定义了 setSquare()函数，传递给它的是我们要更新的当前格子的编号。我们传递的是循环变量 i。当函数接收这一数据时，数据保存在参数 thisSquare 中。可以这样理解：将 i 传递给函数，用这一数据填充参数 thisSquare，但是函数实际上看不到 i。在函数内部，它只知道 thisSquare。

3. var currSquare = "square" +thisSquare;

为了使脚本后面的 getElementById()调用更清晰，我们创建并设置一个新变量：currSquare。这是要处理的当前格子的 id。它将文本字符串"square"和 thisSquare 变量拼接起来。

4. document.getElementById(currSquare).innerHTML = newNum;

这一行找到具有 currSquare 指定的名字的元素，并让它显示 newNum。

3.3 探测对象

在编写脚本时，你可能希望检查浏览器是否有能力理解你要使用的对象。进行这种检查的方法称为对象探测（object detection）。

方法是对要寻找的对象进行条件测试，如下所示：

```
if (document.getElementById) {
```

如果对象存在，if 语句就为 true，脚本继续执行。但是，如果浏览器不理解这个对象，测试就返回 false，并执行条件语句的 else 部分。脚本 3-5 给出所需的 JavaScript，图 3-3 显示在老式浏览器中的结果。

脚本 3-5 对象探测是脚本开发人员的重要工具

```
window.onload = initAll;

function initAll() {
     if (document.getElementById) {
        for (var i=0; i<24; i++) {
           setSquare(i);
        }
     }
     else {
        alert("Sorry, your browser doesn't support this script");
     }
}

function setSquare(thisSquare) {
```

```
        var currSquare = "square" + thisSquare;
        var newNum = Math.floor(Math.random() * 75) + 1;

        document.getElementById(currSquare).innerHTML = newNum;
    }
```



图 3-3　对象探测拒绝这种老式浏览器（Mac 的 Netscape 4）并显示这个错误消息

⇛ 探测对象

1. `if (document.getElementById) {`

这是条件语句的开头。如果圆括号中的对象存在，测试就会返回 `true`，并运行 `initAll()`函数中的正常代码。

2. else {

```
    alert("Sorry, your browser doesn't support this script");
  }
```

如果步骤 1 中的测试返回 `false`，这一行就会弹出警告框，脚本结束运行。

在生产环境中，更好的方法是让用户有别的选择，或者至少提供不需要这一功能的其他页面版本。但是，这里没什么可做的。

✔提示

- 一定要知道，不必总是检查 `document.getElementById`。要检查哪些对象取决于脚本要使用的对象。如果脚本使用的对象并没有得到浏览器 100%的支持，那么总是应该首先检查浏览器是否能够处理它，而不要想当然地认为浏览器可以处理它。为了节省篇幅，本书中的脚本没有进行对象探测，但是在真实环境中，这是很重要的。

已过时的探测方式

对于检查浏览器支持哪些对象，另一种替代方法是进行**浏览器探测**（browser detection），这种方法尝试查明用户使用哪种浏览器查看页面。它向浏览器请求用户代理字符串，这个字符串会报告浏览器名称和版本。然后就可以让脚本以一种方式为某些浏览器服务，而对其他浏览器采用另一种方式。这是一种已经过时的脚本编程方法，因为它的效果不太好。

浏览器探测要求你了解哪些浏览器支持你编写的脚本，哪些不支持。但是，对于你从来没有使用过的浏览器，或者在脚本编写好之后发布的新浏览器，应该怎么办呢？

更糟糕的是，许多浏览器故意报告错误的信息，从而试图通过浏览器探测的检查。例如，苹果的 Safari 浏览器声称它是 Mozilla 浏览器，但它实际上不是。而且大部分浏览器（比如 Safari、Chrome 和 Opera）允许用户设置用户代理字符串。

况且，不断修改脚本来适应所有的浏览器版本不现实，这是一场必输无疑的比赛。

试图探测浏览器支持的 JavaScript 版本也有同样的问题。我们强烈建议不要采用这些探测方法，而是使用对象探测。

3.4 处理数组

在这个示例中，我们要介绍另一个有用的 JavaScript 对象：数组（array）。数组是一种可以存储一组信息的变量。与变量一样，数组可以包含任何类型的数据：文本字符串、数字、其他 JavaScript 对象。在声明数组时，将数组的元素放在圆括号中，以逗号分隔，如下所示：

```
var newCars = new Array("Toyota", "Honda", "Nissan");
```

在此之后，`newCars` 数组就包含这 3 个表示汽车制造商的文本字符串。要访问数组的内容，应该使用变量名和数组成员的索引号（index number），索引号要放在方括号中。`newCars[2]`返回`"Nissan"`，因为与 JavaScript 中的大多数编号一样，数组编号是从零开始的。注意在上面的例子中，我们使用文本字符串作为数组元素。每个文本字符串用直引号括起来，用来分隔数组元素的逗号放在直引号外面。

在这个示例中（见脚本 3-6），我们开始确保 Bingo 卡片是有效的。在真实的 Bingo 卡片上，每列具有不同的数字范围：B 列是 1 ~ 15，I 列是 16 ~ 30，N 列是 31 ~ 45，G 列是 46 ~ 60，O 列是 61 ~ 75。如果再看看图 3-1，就会发现它不是有效的卡片，因为生成它的脚本仅仅是将 1 ~ 75 的随机数简单地放在每个格子里。这个示例只用 3 行新代码纠正这个问题。完成之后，卡片会像图 3-4 这样。它仍然不是有效的 Bingo 卡片（注意，一些列中有重复的数字），但是已经比较接近了。

脚本 3-6 这个脚本限制了每一列中值的范围

```
window.onload = initAll;

function initAll() {
     if (document.getElementById) {
        for (var i=0; i<24; i++) {
           setSquare(i);
        }
     }
     else {
        alert("Sorry, your browser doesn't support this script");
     }
}

function setSquare(thisSquare) {
     var currSquare = "square" + thisSquare;
     var colPlace = new Array(0,0,0,0,0,1,1,1,1,1,2,2,2,2,3,3,3,3,3,4,4,4,4,4);
     var colBasis = colPlace[thisSquare] * 15;
     var newNum = colBasis + Math.floor(Math.random() * 15) + 1;

     document.getElementById(currSquare) innerHTML = newNum;
}
```

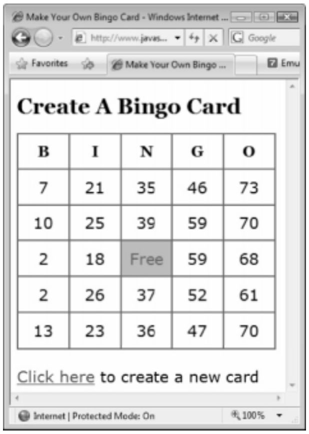


图 3-4　这个 Bingo 卡片已经有所改进，但是仍然不完全有效，因为一些列中有重复的数字

⇒ 使用数组

1. `var colPlace = new Array(0,0,0,0,0,1,1,1,1,1,2,2,2,2,3,3,3,3,3,4,4,4,4,4);`

我们希望限制哪些随机数可以放在哪一列中。最简单的方法是给每一列分配一个编号（B：0，I：1，N：2，G：3，O：4），然后用以下表达式计算可以放进每一列中的数字：（列号×15）+（1～15 的随机数）。

要用 `colPlace` 数组记录每个格子所属的列。它包含 0～4 的数字并重复 5 次（要减去空的格子。请注意，数字 2 仅用了 4 次）。

2.
```
var colBasis = colPlace[thisSquare] * 15;
var newNum = colBasis + Math.floor(Math.random() * 15) + 1;
```

我们首先计算列号：存储在 `colPlace[thisSquare]`中的数字乘以 15。`newNum` 变量仍然生成随机数，但不是 1～75 的数字，而是计算一个 1～15 的随机数，然后加上列号。因此，如果随机数是 7，那么它在 B 列中是 7，在 I 列中是 22，在 N 列中是 37，在 G 列中是 52，在 O 列中是 67。

3.5　处理有返回值的函数

到目前为止，你看到的所有函数都只是做某些事情，然后返回。但有时候，希望函数返回某种结果。脚本 3-7 将前面示例中的一些计算转移到一个函数中，这个函数为 Bingo 卡片上的单元格返回随机数，然后另一个函数使用它返回的结果，这使整个脚本更容易理解了。

脚本 3-7　函数可以返回一个值，然后脚本可以使用这个值

```
window.onload = initAll;

function initAll() {
     if (document.getElementById) {
        for (var i=0; i<24; i++) {
           setSquare(i);
        }
```

```
        }
        else {
            alert("Sorry, your browser doesn't support this script");
        }
}

function setSquare(thisSquare) {
        var currSquare = "square" + thisSquare;
        var colPlace = new Array(0,0,0,0,0,1,1,1,1,1,2,2,2,2,3,3,3,3,3,4,4,4,4,4);
        var colBasis = colPlace [thisSquare] * 15;
        var newNum = colBasis + getNewNum() + 1;

        document.getElementById(currSquare).innerHTML = newNum;
}

function getNewNum() {
        return Math.floor(Math.random() * 15);
}
```

⇛ 从函数返回一个值

1. `var newNum = colBasis + getNewNum() + 1;`

这一行将 newNum 变量设置为需要的数字，但是已经将随机数的生成转移到了函数 getNewNum()中。通过将这一计算转移到函数中，脚本中的运算过程更容易理解了。

2.
```
function getNewNum() {
    return Math.floor(Math.random()* 15);
  }
```

这行代码计算一个 0 ~ 14 的随机数并返回它。在可以使用变量或数字的任何地方，都可以使用这个函数。

✔提示

- ❑ 可以返回任何值。字符串、布尔值和数字都可以。

3.6 更新数组

如图 3-4 所示，这个 Bingo 卡片脚本还无法确保给定的列中不出现重复的数字。这个示例要纠正这个问题，同时说明数组不必进行初始化并读取，而是可以在运行时声明和设置它们。这会提供很大的灵活性，因为可以在脚本运行时通过计算或函数修改数组中的值。脚本 3-8 演示了具体的做法，其中只有几行新代码。

脚本 3-8 将数组的内容改为存储当前值是一种非常强大的技术

```
window.onload = initAll;
var usedNums = new Array(76);

function initAll() {
        if (document.getElementById) {
            for (var i=0; i<24; i++) {
                setSquare(i);
            }
        }
        else {
```

```
            alert("Sorry, your browser doesn't support this script");
        }
    }

    function setSquare(thisSquare) {
        var currSquare = "square" + thisSquare;
        var colPlace = new Array(0,0,0,0,0,1,1,1,1,1,2,2,2,2,2,3,3,3,3,3,4,4,4,4,4);
        var colBasis = colPlace[thisSquare] * 15;
        var newNum = colBasis + getNewNum() + 1;

        if (!usedNums[newNum]) {
            usedNums[newNum] = true;
            document.getElementById(currSquare).innerHTML = newNum;
        }
    }

    function getNewNum() {
        return Math.floor(Math.random() * 15);
    }
```

⇛ 在运行时更新数组

1. `var usedNums = new Array(76);`

这是一种声明数组的新方法。我们将 usedNums 变量声明为一个包含 76 个对象的新数组。正如前面提到的，这些对象可以是任何东西。在这个示例中，它们是布尔值，即 true/false 值。

2. `if (!usedNums[newNum]) {`
    `usedNums[newNum] = true;`

如果 usedNums 数组中的 newNum 位置上是 false（表达式前面的!表示“非”），那么就将它设置为 true，并将 newNum 写到卡片上。如果 newNum 位置上是 true，就什么也不做。这样就不会有重复的数字，但是卡片上可能会留下空格（见图 3-5）。在下一节中，我们要解决这个缺陷。



图 3-5　我们消除了重复的数字，但是现在留下了一些空格子，还需要重新来过

✔提示

- ❑ 为什么这个数组要包含 76 个元素？因为我们希望使用 1 ~ 75 的值。如果将它声明为包含 75 个元素，那么编号是从 0 ~ 74。76 个元素允许我们使用 1 ~ 75 的值，只需忽略第 0 号元素即可。
- ❑ 如果不对布尔值进行初始化，那么它们会自动设置为 false。

3.7　使用 do/while 循环

有时候，需要让代码循环许多次，但是无法确定需要循环多少次。在这种情况下，就要使用 do/while 循环：只要某个条件为 true，就执行某种操作。脚本 3-9 像前面一样写每行的数字，但是这一次在将数字放进单元格之前，它首先检查是否已经使用了这个数字。如果这个数字已经用过了，脚本就生成新的随机数并重复这个过程，直到找到不重复的数字为止。图 3-6 显示了最终有效的 Bingo 卡片。



图 3-6　最终，我们得到了有效的 Bingo 卡片

脚本 3-9　这个脚本防止给定的列中出现重复的数字

```
window.onload = initAll;
var usedNums = new Array(76);

function initAll() {
     if (document.getElementById) {
        for (var i=0; i<24; i++) {
           setSquare(i);
        }
     }
     else {
        alert("Sorry, your browser doesn't support this script");
     }
}

function setSquare(thisSquare) {
     var currSquare = "square" + thisSquare;
     var colPlace = new Array(0,0,0,0,0,1,1,1,1,1,2,2,2,2,3,3,3,3,3,4,4,4,4,4);
     var colBasis = colPlace[thisSquare] * 15;
     var newNum;

     do {
        newNum = colBasis + getNewNum() + 1;
     }
     while (usedNums[newNum]);

     usedNums[newNum] = true;
     document.getElementById(currSquare).innerHTML = newNum;
}
```

```
function getNewNum() {
     return Math.floor(Math.random() * 15);
}
```

⇛ 使用 do/while 循环

1. var newNum;

在前面的示例中，我们在创建 newNum 变量时对它进行初始化。因为我们将多次设置它，所以在进入循环之前创建它，这样只需创建一次。

2. do {

这一行开始 do 代码块。关于这种循环必须记住的一点是，do 块中的代码至少会执行一次。

3. newNum = colBasis + getNewNum() + 1;

与前面的示例一样，循环中的这行代码将 newNum 变量设置为我们需要的数字。

4. }

右花括号结束 do 块。

5. while (usedNums[newNum]);

while 检查会使 do 代码块反复执行，直到检查结果为 false 为止。在这个示例中，我们检查 usedNums[]数组中 newNum 位置上的值，从而检查 newNum 是否已经使用过了。如果这个数字已经用过了，控制就被传递回 do 块的开头，整个过程再次重复。最终，我们会找到一个没有使用过的数字。在此之后，就会离开循环，将 usedNums 设置为 true，并将 newNum 写到卡片上。

✓提示

- do/while 循环的一种常见用途是，从用户输入的数据中去掉空格或无效的字符。但是要记住，无论 while 检查计算出 true 还是 false，do 代码块至少会执行一次。

3.8　以多种方式调用脚本

到目前为止，你看到的脚本都是在加载页面时自动运行的。但是在现实环境中，常常希望让用户对脚本有更多的控制能力，甚至允许他们控制脚本在何时运行。在这个示例中（见脚本 3-10），脚本仍然在加载页面时运行。但是，还允许用户单击页面底部的链接来重新运行脚本，这样就可以完全在浏览器中生成 Bingo 卡片，而不需要从服务器重新加载页面。这向用户提供了快速的响应，而且不会产生服务器负载。

脚本 3-10　让用户有能力自己运行脚本

```
window.onload = initAll;
var usedNums = new Array(76);

function initAll() {
     if (document.getElementById) {
        document.getElementById("reload").onclick = anotherCard;
        newCard();
     }
     else {
        alert("Sorry, your browser doesn't support this script");
     }
}
```

```
function newCard() {
     for (var i=0; i<24; i++) {
        setSquare(i);
     }
}

function setSquare(thisSquare) {
     var currSquare = "square" + thisSquare;
     var colPlace = new Array(0,0,0,0,0,1,1,1,1,1,2,2,2,2,3,3,3,3,3,4,4,4,4,4);
     var colBasis = colPlace[thisSquare] * 15;
     var newNum;

     do {
        newNum = colBasis + getNewNum() + 1;
     }
     while (usedNums[newNum]);

     usedNums[newNum] = true;
     document.getElementById(currSquare).innerHTML = newNum;
}

function getNewNum() {
     return Math.floor(Math.random() * 15);
}

function anotherCard() {
     for (var i=1; i<usedNums.length; i++) {
        usedNums[i] = false;
     }

     newCard();
     return false;
}
```

⇛ 以多种方式调用脚本

1. `document.getElementById("reload").onclick = anotherCard;`
   `newCard();`

我们已经看到过 initAll()函数，但是这里的 initAll()函数有一些变化：在 HTML 页面（它的 id 是 reload，见脚本 3-1）上设置链接，让它在被单击时调用 anotherCard()。initAll()函数中原来的所有计算现在被转移到新的 newCard()函数中，这里只需调用 newCard()函数，newCard()函数中没有需要讨论的新代码。

2.
```
function anotherCard() {
    for (var i=1; i<usedNums.length; i++) {
      usedNums[i] = false;
    }

    newCard();
    return false;
  }
```

这是 anotherCard()函数，当用户单击链接时会调用这个函数。它执行 3 个操作：

- 将 usedNums[]数组中的所有元素设置为 false（这样就可以重新使用所有数字）;
- 调用 newCard()函数（生成另一个卡片）;
- 返回 false 值，使浏览器不尝试加载链接的 href 中指定的页面（这在第 2 章中讨论过）。

✓提示

- 到了现在，你已经知道了如何使用 JavaScript 重新加载页面的一部分，而不需要向服务器请求整个新页面，而这正是许多人认为的 Ajax 基本特色。从第 13 章起，我们将详细讨论 Ajax。

3.9 组合使用 JavaScript 和 CSS

对于目前的 Bingo 示例，你可能会觉得奇怪："他们都说 JavaScript 可以提供所有交互功能，但是为什么看不到任何用户交互呢？"这是一个很合理的问题，现在就讲解如何让用户能够操作所生成的 Bingo 卡片。为此，脚本 3-11 通过添加一些 JavaScript 来利用 CSS 的功能。

脚本 3-11　通过 JavaScript 添加一个类，使代码可以利用 CSS 的功能

```
window.onload = initAll;
var usedNums = new Array(76);

function initAll() {
     if (document.getElementById) {
        document.getElementById("reload").onclick = anotherCard;
        newCard();
     }
     else {
        alert("Sorry, your browser doesn't support this script");
     }
}

function newCard() {
     for (var i=0; i<24; i++) {
        setSquare(i);
     }
}

function setSquare(thisSquare) {
     var currSquare = "square" + thisSquare;
     var colPlace = new Array(0,0,0,0,0,1,1,1,1,1,2,2,2,2,3,3,3,3,3,4,4,4,4,4);
     var colBasis = colPlace[thisSquare] * 15;
     var newNum;

     do {
        newNum = colBasis + getNewNum() + 1;
     }
     while (usedNums[newNum]);

     usedNums[newNum] = true;
     document.getElementById(currSquare).innerHTML = newNum;
     document.getElementById(currSquare).className = "";
     document.getElementById(currSquare).onmousedown = toggleColor;
}

function getNewNum() {
```

```
        return Math.floor(Math.random() * 15);
    }

    function anotherCard() {
        for (var i=1; i<usedNums.length; i++) {
            usedNums[i] = false;
        }

        newCard();
        return false;
    }

    function toggleColor(evt) {
        if (evt) {
            var thisSquare = evt.target;
        }
        else {
            var thisSquare = window.event.srcElement;
        }
        if (thisSquare.className == "") {
            thisSquare.className = "pickedBG";
        }
        else {
            thisSquare.className = "";
        }
    }
```


⇛ 使用 JavaScript 应用样式

1. `document.getElementById(currSquare).className = "";`
   `document.getElementById(currSquare).onmousedown = toggleColor;`

因为 Bingo 卡片是可以重复使用的，所以要确保 Bingo 卡片最初是空的：对于在 setSquare()中设置的每个格子，把 class 属性设置为""（空字符串），并让 onmousedown 事件处理程序调用新的 toggleColor()函数。

2. `function toggleColor(evt) {`

如果你是 CSS 专家，那么可能会注意到，脚本 3-2 中声明了我们从来没有使用过的样式。现在，在新的 toggleColor()函数中就要使用这些样式了。用户现在可以单击卡片上的任何格子，这个格子的背景颜色就会改变，表示这个号码已经被叫过了。

3.
```
if (evt) {
    var thisSquare = evt.target;
}
else {
    var thisSquare = window.event.srcElement;
}
```

首先，需要查明单击的是哪个格子。但糟糕的是，这有两种方式：IE 方式和其他所有浏览器处理事件的方式。

如果一个称为 evt 的值被传递给这个函数，就说明用户的浏览器不是 IE，可以看到 evt 的目标。如果浏览器是 IE，就需要查看 window 对象的 event 属性的 srcElement 属性。无论采用哪种方式，都会得到 thisSquare 对象，然后可以检查和修改这个对象。

```
4. if (thisSquare.className == "") {
     thisSquare.className = "pickedBG";
   }
   else {
     thisSquare.className = "";
   }
```

在这里，检查被单击的格子的 class 属性是否有一个值。如果没有，就给它设置 pickedBG 值，因为格子的背景表示这个数字已经被选择了。

现在仅仅修改 class 属性并不会改变页面上的任何东西，但是请回顾脚本 3-2 中的 CSS。class 为 pickedBG 的任何标签的背景颜色与 free 格子相同。在这里修改 class 会对这个格子自动应用这个样式，使它显示粉红色背景（见图 3-7）。

当然，用户可能选错格子，需要有办法纠正错误。再次单击格子，这一次 className 有一个值，所以把它恢复为空字符串。

也可以不修改格子上的 class 属性，而是修改它的 style 属性，这样就不必为 CSS 文件操心了。但是这种方法不好。我们希望利用 CSS 文件，因为它只修改页面的可视外观，而不修改页面的行为。

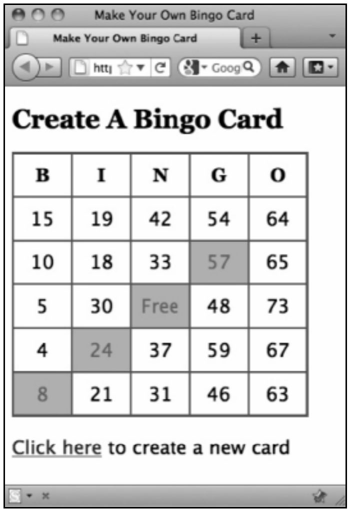


图 3-7　用户叫过号码的格子会被加上标记

关于位的说明

在使用布尔值时，处理的值要么是 true，要么是 false。也可以认为这些变量包含 0 或 1，因为这是计算机在内部处理任何信息的方式（把 true 表示为 1，把 false 表示为 0）。

这些值（0 和 1）称为位（bit）。这些位组成计算机存储的信息。可以把每个位看作一个电灯开关，其状态要么是开，要么是关。

因为计算机中的所有信息只是一系列位，所以需要能够操作这些位。尤其是需要能够比较它们。下面是需要的一些内部操作。

- 与（&）：在对两个位进行“与”计算时，如果它们都是 true（即都是 1），结果就是 true；否则，结果是 false。
- 或（|）：在对两个位进行“或”计算时，如果其中之一是 true（即其中之一是 1），结果就是 true；如果它们都是 false，结果就是 false。

对大于 1 的数字使用“与”和“或”操作符，称为**位算术**（bitwise arithmetic）。计算机在内部把每个数字转换为二进制值，然后比较各个位。因为这个过程在内部进行，所以不需要你亲自执行转换。

3.10　检查状态

让用户能够给格子加标记之后，还可以检查格子是否构成了获胜模式。在这个示例中，用户选择那些已经被叫过的号码，脚本 3-12 会让用户知道他们什么时候获胜了。

脚本 3-12　这个脚本使用复杂的数学计算判断获胜组合


```
window.onload = initAll;
var usedNums = new Array(76);

function initAll() {
     if (document.getElementById) {
        document.getElementById("reload").onclick = anotherCard;
        newCard();
     }
     else {
        alert("Sorry, your browser doesn't support this script");
     }
}

function newCard() {
     for (var i=0; i<24; i++) {
        setSquare(i);
     }
}

function setSquare(thisSquare) {
     var currSquare = "square" + thisSquare;
     var colPlace = new Array(0,0,0,0,0,1,1,1,1,1,2,2,2,2,3,3,3,3,3,4,4,4,4,4);
     var colBasis = colPlace [thisSquare] * 15;
     var newNum;

     do {
        newNum = colBasis + getNewNum() + 1;
     }
     while (usedNums[newNum]);

     usedNums[newNum] = true;
     document.getElementById(currSquare).innerHTML = newNum;
     document.getElementById(currSquare).className = "";
     document.getElementById(currSquare).onmousedown = toggleColor;
}

function getNewNum() {
     return Math.floor(Math.random() * 15);
}

function anotherCard() {
     for (var i=1; i<usedNums.length; i++) {
        usedNums[i] = false;
     }

     newCard();
     return false;
}

function toggleColor(evt) {
     if (evt) {
        var thisSquare = evt.target;
```

```
    }
    else {
       var thisSquare = window.event.srcElement;
    }
    if (thisSquare.className == "") {
       thisSquare.className = "pickedBG";
    }
    else {
       thisSquare.className = "";
    }
    checkWin();
}

function checkWin() {
    var winningOption = -1;
    var setSquares = 0;
    var winners = new Array(31,992,15360,507904,541729,557328,1083458,2162820,4329736,8519745,8659472,
    ➝16252928);

    for (var i=0; i<24; i++) {
       var currSquare = "square" + i;
       if (document.getElementById(currSquare).className != "") {
          document.getElementById(currSquare).className = "pickedBG";
          setSquares = setSquares | Math.pow(2,i);
       }
    }

    for (var i=0; i<winners.length; i++) {
       if ((winners[i] & setSquares) == winners[i]) {
          winningOption = i;
       }
    }

    if (winningOption > -1) {
       for (var i=0; i<24; i++) {
          if (winners[winningOption] & Math.pow(2,i)) {
             currSquare = "square" + i;
             document.getElementById(currSquare).className = "winningBG";
          }
       }
    }
}
```

这个示例要使用一些比较复杂的数学计算。如果你以前没有接触过二进制运算，可以先阅读前面的补充内容“关于位的说明”。如果希望深入了解，可以阅读后面的补充内容 “按位运算详解”（如果认为不需要了解这些细节，也可以跳过相关内容）。

⇛ 检查获胜状态

1. `checkWin();`

每当用户给格子加标记时，获胜状态都可能会有变化，所以在 `toggleColor()`的末尾要调用 `checkWin()`。

2.
```
var winningOption = -1;
var setSquares = 0;
var winners = new Array(31, 992, 15360, 507904, 541729, 557328, 1083458, 2162820, 4329736,
➝8519745, 8659472, 16252928);
```

在 `checkWin()`的开头创建下面这些新变量。

- ❑ winningOption，存储用户可能遇到的获胜选项（如果有的话）。
- ❑ setSquares，存储已经单击的格子。
- ❑ winners，是一个数字数组，其中的每个数字是有效获胜组合的编码值。

3.
```
for (var i=0; i<24; i++) {
   var currSquare = "square" + i;
   if (document.getElementById(currSquare).className != "") {
```

对于卡片上的每个格子，需要检查它的号码是否已经被叫过。我们将使用格子的 class 属性作为标志——如果它是空的，就是没有被单击过；如果有 class 属性值，就执行下面的代码。

4.
```
document.
getElementById(currSquare).className = "pickedBG";
setSquares = setSquares | Math.pow(2,i);
```

第一行很简单，实际上它应该是多余的——class 属性已经设置为了 pickedBG。但是，也可能有例外，比如当用户单击了他们原本不打算单击的格子时，碰巧获胜了（这会把 class 属性重新设置为 winningBG 而不是 pickedBG），然后用户再次单击这个格子恢复它的状态。如果用户实际上是获胜者，后面会重新设置 class 属性。

第二行使用*按位算术*，根据卡片的每个可能状态把 setSquares 设置为一个数字。单竖杠（|）对两个值执行位“或”计算：setSquares 本身和数字 2^i。2^i 是 Math.pow(2,i)的结果，2^0 是 1，2^1 是 2，2^2 是 4，依次类推。对这两个数字执行按位“或”操作会产生一个表示用户所处状态的变量（共有超过 1600 万个可能的状态）。

5.
```
for (var i=0; i<winners.length; i++) {
   if ((winners[i] & setSquares) == winners[i]) {
      winningOption = i;
   }
}
```

这是第二个比较复杂的部分。现在只知道卡片当前所处的状态，我们希望查明它是否是获胜状态。在一般的 Bingo 游戏中，有 12 个获胜状态，这部分代码把卡片的当前状态与每个获胜状态作比较。我们在每个获胜状态和当前状态之间执行按位“与”计算。这会产生一个新状态，在这个状态中只有那些同时出现在两个状态中的格子，才具有 true 值。然后，对这个新状态和同一个获胜状态作比较，这样就能够判断卡片是否完全符合这个获胜模式，即结果不包含获胜状态之外的所有选择（因为在获胜状态中找不到它们），而且只要在获胜模式中找到的所有格子也出现在当前模式中，用户就是获胜者。在这里，把 winningOption 设置为 i，这就是用户匹配的模式。

6.
```
if (winningOption > -1) {
   for (var i=0; i<24; i++) {
      if (winners[winningOption] & Math.pow (2,i)) {
         currSquare = "square" + i;
         document.getElementById(currSquare).className = "winningBG";
      }
   }
}
```

最后，如果 `winningOption` 是大于-1 的数字，就说明用户是获胜者。在这种情况下，希望循环遍历每个格子，检查在获胜模式中是否可以找到它。如果是，就把 `class` 属性设置为 `winningBG`，我们的工作就完成了，见图 3-8。



图 3-8　看起来找到获胜者了（颜色是闪烁的，但是在印刷的书中看不出来）

✔提示

- 同样，只需设置获胜格子的 `class` 属性，让它与某个 CSS 样式匹配，就能够改变卡片的外观。但是，因为纸张是静态的，所以看不出实际的效果：`winningBG` 规则把背景设置为一个动画 gif，这张图片会在红色和白色之间缓慢地变化。我的一个朋友称它为“可爱的讨厌鬼”。
- 有许多种规则不同的 Bingo 游戏，它们的获胜模式各不相同。只需修改 `winners` 数组的初始化代码，脚本就能够适应任何 Bingo 游戏。
- 如果你以前对于条件表达式为什么使用`&&`和`||`（分别表示“与”和“或”）觉得奇怪，现在应该明白原因了：JavaScript 使用这些操作符的单字符版本执行二进制算术而非十进制。

按位运算详解

（正如前面提到的，这里讲解的内容很复杂，在开始阅读之前请作好心理准备。如果还没有作好准备，请跳过这部分。）

既然只有 12 种可能的获胜模式（至少在这个版本的 Bingo 游戏中是这样），我们可以很容易地编写代码检查每个获胜模式。但是 Bingo 游戏有许多不同的版本，这意味着如果希望玩另一个版本，就必须彻底修改检查获胜者的方式。

为了避免这个问题，我们使用按位算术（参见“关于位的说明”）来存储获胜模式。这意味着，我们要利用计算机在内部以 1 和 0 记录所有信息这一性质。所以，我们创建一个列表的程序形式：

```
0    √     1
1          2
2          4
3          8
```

```
4              16
5     √        32
6              64
7             128
8             256
9             512
10    √      1024
11           2048
12           4096
13           8192
14    √     16384
15          32768
16          65536
17         131072
18         262144
19    √    524288
20        1048576
21        2097152
22        4194304
23        8388608
```

如果比较一下对钩左边列中的数字，就会发现我们选择了卡片最上面的一行：square0、square5、square10、square14 和 square19。右边列中的数字是以左边数字作为指数的 2 的幂。

要想得到获胜模式的数字形式，只需找到这个模式包含的格子，并把它们右边的数字加起来。在这里，就是 1+32+1024+16384+524288=541729，这就是在获胜者列表中包含的数字。

要想计算 B 列中所有格子的数字形式，只需计算 1+2+4+8+16，结果是 31。这是另一个获胜者。对于每个获胜模式，都这样做。

下面到了关键之处。如果把上面列表的次序颠倒过来，即从 23 到 0，然后把对钩替换为 1，空的地方替换为 0，就会得到一个 24 位的二进制数字。因此，第一个获胜模式可以看作 000010000100010000100001，第二个获胜模式是 000000000000000000011111。前者是 541729 的二进制形式，后者是 31 的二进制形式。

在检查卡片上的每个格子是否被设置时，把结果存储在 setSquares 中。这个值是玩家选择的所有格子的总和。同样，可以把它看作一个 24 位的二进制数字。

对所有这些值执行“或”计算，得到 setSquares。在对两个 0 执行“或”计算（使用单竖杠|）时，结果是 0。对于其他任何组合，结果都是 1。

假设最终的 setSquares 是 561424，这意味着玩家选择了格子 4、8、12、15 和 19，二进制值是 000010001001000100010000。现在，561424 不在获胜者列表中。但是，对这个数字和 557328（一个获胜者）执行“与”计算，会得到：

```
000010001001000100010000 &
000010001000000100010000
000010001000000100010000
```

在对两个 1 执行“与”计算（使用一个&符号）时，结果是 1。对于其他任何组合，结果都是 0。

在代码中，我们再把结果值与获胜模式值作比较，如果它们是相同的（就像这里的情况），我们就找到了获胜者。在这里，这是从左下角到右上角的对角线组合。

如果你现在还怀疑这样做是否真的比手工计算简便，请考虑：如果选择所有 4 个角上的格子（即格子 0、4、19 和 23）也算是获胜，那么只需把数字 8912913 添加到获胜者数组中，其他地方都不需要修改。

3.11　处理字符串数组

到目前为止，我们使用的所有数组都是由布尔值或数字组成的。作为最后一个与 Bingo 相关的示例，脚本 3-13 用一个字符串数组综合使用了前面学习的所有技术，创建一个流行的“Buzzword Bingo”游戏。

脚本 3-13　这个 Buzzword Bingo 游戏可以在沉闷的员工会议上活跃气氛，你只需添加自己的文本字符串

```
var buzzwords = new Array ("Aggregate",
     "Ajax",
     "API",
     "Bandwidth",
     "Beta",
     "Bleeding edge",
     "Convergence",
     "Design pattern",
     "Disruptive",
     "DRM",
     "Enterprise",
     "Facilitate",
     "Folksonomy",
     "Framework",
     "Impact",
     "Innovate",
     "Long tail",
     "Mashup",
     "Microformats",
     "Mobile",
     "Monetize",
     "Open social",
     "Paradigm",
     "Podcast",
     "Proactive",
     "Rails",
     "Scalable",
     "Social bookmarks",
     "Social graph",
     "Social software",
     "Spam",
     "Synergy",
     "Tagging",
     "Tipping point",
     "Truthiness",
     "User-generated",
     "Vlog",
     "Webinar",
     "Wiki",
     "Workflow"
);
```

```
var usedWords = new Array(buzzwords.length);
window.onload = initAll;

function initAll() {
     if (document.getElementById) {
        document.getElementById("reload").onclick = anotherCard;
        newCard();
     }
     else {
        alert("Sorry, your browser doesn't support this script");
     }
}

function newCard() {
     for (var i=0; i<24; i++) {
        setSquare(i);
     }
}

function setSquare(thisSquare) {
     do {
        var randomWord = Math.floor(Math.random() * buzzwords.length);
     }
     while (usedWords[randomWord]);

     usedWords[randomWord] = true;
     var currSquare = "square" + thisSquare;
     document.getElementById(currSquare).innerHTML = buzzwords[randomWord];
     document.getElementById(currSquare).className = "";
     document.getElementById(currSquare).onmousedown = toggleColor;
}

function anotherCard() {
     for (var i=0; i<buzzwords.length; i++) {
        usedWords[i] = false;
     }

     newCard();
     return false;
}

function toggleColor(evt) {
     if (evt) {
        var thisSquare = evt.target;
     }
     else {
        var thisSquare = window.event.srcElement;
     }
     if (thisSquare.className == "") {
        thisSquare.className = "pickedBG";
     }
     else {
        thisSquare.className = "";
     }
     checkWin();
}

function checkWin() {
     var winningOption = -1;
     var setSquares = 0;
```

3

```
    var winners = new Array(31,992,15360,507904,541729,557328,1083458,2162820,4329736,8519745,8659472,
    ➝16252928);

    for (var i=0; i<24; i++) {
       var currSquare = "square" + i;
       if (document.getElementById(currSquare).className != "") {
          document.getElementById(currSquare).className = "pickedBG";
          setSquares = setSquares | Math.pow(2,i);
       }
    }

    for (var i=0; i<winners.length; i++) {
       if ((winners[i] & setSquares) == winners[i]) {
          winningOption = i;
       }
    }

    if (winningOption > -1) {
       for (var i=0; i<24; i++) {
          if (winners[winningOption] & Math.pow(2,i)) {
             currSquare = "square" + i;
             document.getElementById(currSquare).className = "winningBG";
          }
       }
    }
}
```

⇛ 使用字符串数组

1.
```
var buzzwords = new Array("Aggregate", "Ajax", "API", "Bandwidth", "Beta", "Bleeding edge",
➝"Convergence", "Design pattern", "Disruptive", "DRM", "Enterprise", "Facilitate",
➝"Folksonomy", "Framework","Impact", "Innovate", "Long tail", "Mashup", "Microformats",
➝"Mobile", "Monetize", "Open social", "Paradigm", "Podcast", "Proactive", "Rails", "Scalable",
➝"Social bookmarks", "Social graph", "Social software", "Spam", "Synergy", "Tagging",
➝"Tipping point", "Truthiness", "User-generated", "Vlog", "Webinar", "Wiki", "Workflow");

var usedWords = new Array(buzzwords.length);
```

这个 Buzzword Bingo 游戏使用的字符串都与“Web 2.0”有关，但是你可以在 buzzwords 数组中添加关于任何主题的字符串。需要至少 24 个字符串（越多越好），而且字符串不应该太长（否则在格子中放不下）。除此之外没有任何其他限制，你可以尽量发挥想象力。

在初始化字符串数组时，还需要初始化一个新的布尔值数组 usedWords。把它的大小声明为 buzzwords.length，这样在添加新字符串时就不需要修改 usedWords 数组——它会自动地调整到合适的长度。

2.
```
do {
   var randomWord = Math.floor(Math.random() * buzzwords.length);
}
while (usedWords[randomWord]);

usedWords[randomWord] = true;
var currSquare = "square" + thisSquare;
document.getElementById(currSquare).innerHTML = buzzwords[randomWord];
```

决定把哪个字符串放在哪个格子中实际上更简单，因为任何字符串可以放在任何格子中（没有标准 Bingo 游戏中的编号限制）。只需确保获取一个还未使用过的字符串，把它标为已使用，然后写到格子中。

3.
```
for (var i=0; i<buzzwords.length; i++) {
   usedWords[i] = false;
}
```

与标准 Bingo 卡片一样，在生成一张新卡片时，必须把 usedWords 中的所有标志设置为 false，这样就可以再次使用它们。

✔提示

- 在旧金山举行的年度 Macworld 博览会上，通常由 Steve Jobs 致开幕词（因此这被称为 Steve Note）。另一个传统是与会者要玩 SteveNote Bingo，也就是猜猜 Steve 会说哪些他喜欢说的词，比如“Boom!”和“One more thing...”）以及哪些谣传的产品真的会出现。

  因为 iPhone 附带了标准的 Safari 浏览器，所以我在 Macworld 上介绍 iPhone 之后展示了这个游戏的交互式版本（见图 3-9）。它非常受欢迎，尽管在致开幕词时没有人真的喊出“Bingo!”。这个 Bingo 示例也可用于其他突发新闻事件，比如直播的美国政治辩论。



图 3-9 只需一个移动浏览器，再加上一点想象力，就可以针对几乎任何场合编写一个 Bingo 游戏

第 4 章

处理图像

JavaScript 最常见也最显著的用途之一是在网页上添加动画，从而在视觉上更具吸引力，这就是本章的主题。用 JavaScript 实现的一种常见且有效的效果是，当用户将鼠标移动到图像上时，会改变网页上的图像，这样页面就能对用户的操作及时作出反应。这种称为翻转器（rollover）的效果很容易实现，而且有许多应用场合。

翻转器是强大的工具，除了翻转器之外，还可以用 JavaScript 实现很多效果，比如自动改变图像、创建广告条、建立幻灯片和在页面上显示随机图像。

在本章中，你将学习如何用 JavaScript 实现所有这些图像效果。让我们开始吧。

目前需要了解的 HTML 知识见表 4-1。

表4-1　目前需要了解的HTML知识——图像

标　　签	属　　性	意　　义
img		其中包含描述浏览器要显示的图像的属性
	src	包含图像的URL，这个URL是相对于网页的URL来说的
	width	包含浏览器显示图像所用的宽度（以像素为单位）
	height	包含浏览器显示图像所用的高度（以像素为单位）
	alt	用来在非图形化浏览器中替代图像

4.1　创建翻转器

翻转器背后的思想很简单。有两个图像。第一个图像是原始（original）图像，它与网页的其他部分一起加载和显示。当用户将鼠标移动到第一个图像上时，浏览器快速地将第一个图像替换为第二个图像，即替换（replacement）图像，这样就产生了运动或动画效果。

脚本 4-1 给出了最基本的翻转器，所有代码都在标准的图像链接中。首先加载蓝色箭头（见图 4-1），当用户将鼠标移动到这个图像上时，用一个红色的箭头替代它（见图 4-2）。当用户移走鼠标时，重新绘制蓝色箭头。

脚本 4-1　这是在链接标签中实现翻转器的最简单方法

```
<!DOCTYPE html>
<html>
<head>
```

```
    <title>A Simple Rollover</title>
    <link rel="stylesheet" href="script01.css">
</head>
<body>
    <a href="next.html" onmouseover="document.images['arrow'].src='images/arrow_on.gif'" onmouseout=
    ➝"document.images['arrow'].src='images/arrow_off.gif'"><img src="images/arrow_off.gif" id="arrow"
    ➝alt="arrow"></a>
</body>
</html>
```

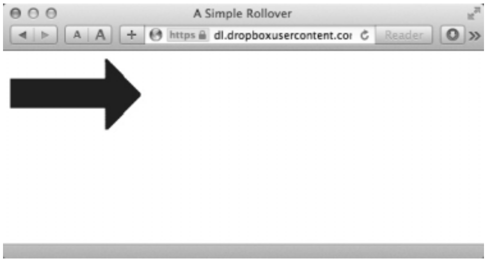



图 4-1　在用户将鼠标移动到图像上之前显示的第一个图像

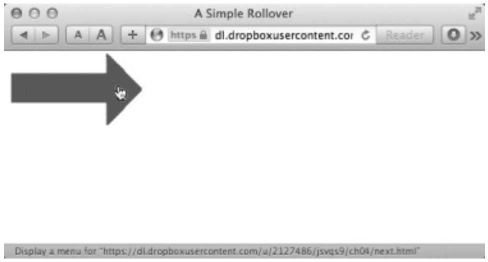


图 4-2　当鼠标移动到图像上时，脚本用第二个图像替换第一个图像

一些样式可以应用于页面上的元素，我们将这些样式单独放在一个 CSS 文件中，如脚本 4-2 所示。

脚本 4-2　在本章的许多示例中，用于样式元素的 CSS 文件

```
body {
    background-color: #FFF;
}

img {
    border-width: 0;
}

img#arrow, img#arrowImg {
    width: 147px;
    height: 82px;
}

#button1, #button2 {
    width: 113px;
    height: 33px;
}

.centered {
    text-align: center;
}

#adBanner {
    width: 400px;
    height: 75px;
}
```

⇒ 创建翻转器

1. `<a href="next.html"`

链接首先指定当用户单击图像时，浏览器将转到哪里，在这个示例中是 next.html 页面。

2. `onmouseover="document.images['arrow'].src='images/arrow_on.gif'"`

当用户将鼠标移动到图像上时（id 为 arrow 的 src），images 目录中的替换图像 arrow_on.gif 被写到文档窗口中。

3. `onmouseout="document.images['arrow'].src='images/arrow_off.gif'">`

当鼠标移走时，arrow_off.gif 图像重新显示。

4. `<img src="images/arrow_off.gif" id="arrow" alt="arrow">`

图像链接为页面定义原始图像的来源。

✔提示

- 我们在图像标签中包含了 alt 属性，这是因为如果希望 HTML 符合 W3C 标准，就必须有 alt 属性（这个属性为非图形化浏览器提供图像的名称或描述），而且使用 alt 属性有助于残障人士访问你的页面，比如用屏幕阅读器浏览页面的盲人用户。
- 确保所有图像的 on 版本都存在，否则一旦用户将鼠标悬停到链接上，页面会显示无效图片图标。
- 示例代码中同时使用了双引号和单引号，你可能想知道两者有何区别。基本上跟英文中的用法一样：在已经使用双引号的语句中再使用引号就需要用单引号。

  除此之外，JavaScript 并不在意开发者使用哪种引号。唯一需要注意的就是引号一定要成对出现，以单引号开始就要以单引号结束，同样，以双引号开始就要以双引号结束。

这种翻转器的缺点

这种实现翻转器的方法非常简单，但是你应该知道它有几个问题和缺点。

- 因为第二个图像是在用户将鼠标移动到第一个图像上时从服务器下载的，所以在第二个图像替换第一个图像之前，可以察觉到延迟，这对于用调制解调器而不是宽带连接浏览站点的用户尤其明显。
- 使用这种方法在老式浏览器中会产生错误消息，比如 Netscape 2.0 或更早的版本、IE 3.0 或更早的版本，或者 America Online 2.7 浏览器。因为很少有人还在使用这些过时的浏览器，所以这已经不是什么大问题了。

我们建议不要使用这种方法，而是使用 4.2 节中介绍的方法来创建翻转器，从而解决这些问题以及其他问题。

4.2　创建更有效的翻转器

为了产生动画的效果，需要确保替换图像立刻出现，而不能有从服务器获得图像所造成的延迟。为此，使用 JavaScript 预先将所有图像加载到浏览器的缓存中（这样，当需要它们时，它们已经在用户的硬盘上了），并且将图像放进脚本使用的变量中。这样的话，当用户将鼠标移动到图像上时，脚本就会用包含替换图像的第二个变量替换包含原图像的变量。脚本 4-3 演示了具体的做法。视觉效果

与图 4-1 和图 4-2 相同，但是动画更加流畅。

脚本 4-3 这个 HTML 页面上唯一的 JavaScript 是对外部 .js 文件的引用

```
<!DOCTYPE html>
<html>
<head>
     <title>A More Effective Rollover</title>
     <script src="script02.js"></script>
     <link rel="stylesheet" href="script01.css">
</head>
<body>
     <a href="next1.html"><img src="images/button1_off.gif" alt="button1" id="button1"></a>  
     <a href="next2.html"><img src="images/button2_off.gif" alt="button2" id="button2"></a>
</body>
</html>
```


为了使 JavaScript 更便于管理，我们将 JavaScript 代码从 HTML 页面中提取出来并将它放进外部 .js 文件中，见脚本 4-4（关于 .js 文件的更多知识见第 2 章）。

脚本 4-4 这是比脚本 4-1 更好的实现翻转器的方法，因为它要灵活得多

```
window.onload = rolloverInit;

function rolloverInit() {
     for (var i=0; i<document.images.length; i++) {
        if (document.images[i].parentNode.tagName == "A") {
           setupRollover(document.images[i]);
        }
     }
}

function setupRollover(theImage) {
     theImage.outImage = new Image();
     theImage.outImage.src = theImage.src;
     theImage.onmouseout = function() {
        this.src = this.outImage.src;
     }

     theImage.overImage = new Image();
     theImage.overImage.src = "images/" + theImage.id + "_on.gif";
     theImage.onmouseover = function() {
        this.src = this.overImage.src;
     }
}
```

⇛ 创建更好的翻转器

1. `<script src="script02.js"></script>`

这个标签在 HTML 页面（见脚本 4-3）中。它使用 `src` 属性向浏览器说明到哪里寻找外部 .js 文件，也就包含 JavaScript 的文件。

2. `<a href="next1.html"><img src="images/button1_off.gif" alt="button1" id="button1"> </a>`
`→  `
`<a href="next2.html"><img src="images/button2_off.gif" alt="button2" id="button2"></a>`

这两个标签仍然在脚本 4-3 中，它们是用于按钮的典型链接标签，其中嵌入了图像标签。`href` 属

性描述当用户单击链接时链接的目标。在 img 标签中，src 属性提供在用户将鼠标移动到图像上之前显示的图像的路径。链接标签也定义了图像的 alt 文本。注意，这两个按钮都有 id 属性。正如第 1 章描述的，id 对于每个对象必须是唯一的。脚本使用图像的 id 使翻转器发挥作用。

3. window.onload = rolloverInit;

现在转到脚本 4-4，当页面完成加载时，会触发 window.onload 事件处理程序。这个处理程序调用 rolloverInit()函数。

这里使用这个处理程序确保在页面完成加载之前脚本不会执行。这是因为在页面完成加载之前，页面上的某些元素如果还没有加载，引用页面上的元素可能会导致错误。

4. function rolloverInit() {

```
    for (var i=0; i<document.images.length; i++) {
```

rolloverInit()函数扫描页面上的每个图像，检查图像外边的标签是否是<a>标签，如果是，就说明它是一个链接。这两行中的第一行是函数的开头。第二行开始一个 for...next 循环，这个循环遍历所有图像。它首先将计数器变量 i 设置为 0。然后，在每次循环迭代中，如果 i 的值小于文档中的图像数量，就将 i 递增 1。

5. if (document.images[i].parentNode.tagName == "A") {

这里检查包围图像的标签是否是锚标签。检查方法是查看对象的值是否为 A（A 是锚标签的名称）。我们来仔细看看这里的表达式。第一部分 document.images[i]是当前的图像。它的 parentNode 属性是包围它的容器标签，而 tagName 提供容器标签的名称。所以，圆括号中代码的意思是："对于这个特定的图像，包围它的标签是 A 吗？"

6. setupRollover(document.images[i]);

如果步骤 5 中的测试结果是 true，那么调用 setupRollover 函数并传递当前图像。

7. function setupRollover(theImage) {

在逐行查看这个函数之前，先看一下整个函数的作用：这个函数将两个新的属性添加到传递给它的图像对象中。新的属性是 outImage（鼠标不在图像上时的图像版本）和 overImage（鼠标在图像上时的图像版本），它们本身都是图像对象。因为它们是图像对象，所以被创建后，就可以添加它们的 src 属性了。outImage 的 src 值是当前图像的 src，overImage 的 src 值是根据原图像的 id 属性计算出来的。

这一行开始 setupRollover 函数，其参数是前面的 rolloverInit()函数传递给它的图像。

8. theImage.outImage = new Image();

这一行获得传递进来的图像对象，并在其中添加新的 outImage 属性。因为可以在对象上添加任何类型的属性，而且属性本身也是对象，所以这里的操作是将一个图像对象添加到图像中。新的图像对象的圆括号是可选的，但这是良好的做法。如果需要的话，可以通过传递参数设置新图像对象的属性。

9. theImage.outImage.src = theImage.src;

现在，将新的 outImage 的来源设置为与 theImage 的来源相同。页面上的默认图像也就是鼠标不在图像上时看到的图像。

10. theImage.onmouseout = function() {

```
    this.src = this.outImage.src;
  }
```

第一行开始定义一个匿名函数（也就是没有函数名的函数）。可以给它指定一个名称（比如 rollOut），但是因为它只有一行代码，所以不需要命名。

在这里，我们告诉浏览器当用户鼠标离开图像时应该触发什么操作。当发生这种情况时，希望把图像源恢复为最初的值，也就是图像的 outImage 版本。

11.
```
theImage.overImage = new Image();
theImage.overImage.src = "images/" + theImage.id + "_on.gif";
```

在第一行中，创建一个新的图像对象，它将包含图像的 overImage 版本。第二行设置 overImage 的来源。它将"images/"和图像的 id（在脚本 4-3 中，我们看到这些 id 是 button1 和 button2）拼接起来，再加上"_on.gif"，从而在运行时构造出源文件名。

12.
```
theImage.onmouseover = function() {
   this.src = this.overImage.src;
}
```

这是另一个匿名函数。它告诉浏览器，当用户将鼠标移动到图像上时，应该把当前图像的源重新设置为 overImage 版本，见图 4-3 和图 4-4。



图 4-3　也可以在同一页面上建立多个翻转器



图 4-4　移动到第二个翻转器上

✔提示

- 在为翻转器准备图像时，要确保所有 GIF 图像或者 PNG 图像都是不透明的。如果它们是透明的，那么会透过透明的图像看到被替换掉的图像，这不是我们需要的效果。
- 原图像和替换图像的尺寸应该相同。否则，一些浏览器会替你重新设置尺寸，而调整后的结果可能不理想。
- 在前面的示例中，当把鼠标移动到链接上时，翻转器会发挥作用。在这里，当把鼠标移动到图像上时，翻转器会发挥作用——也就是说，onmouseover 和 onmouseout 现在附在图像上，而不是链接。尽管这两种方法往往产生同样的效果，但是有一个大的差异：一些比较老式的浏览器（Netscape 4 及更早的版本，IE 3 及更早的版本）在 img 标签上不支持 onmouseover 和 onmouseout。
- 你可能认为，因为 HTML 页面上的所有标签都是小写的，所以 tagName 应该与小写的 a 进行比较。但这不对，tagName 总是返回大写的值。
- 编写翻转器的方式有很多种，我们喜欢使用这个因为它很灵活：向相关 HTML 页面添加图像或者从页面删除图像都不需要改写代码。

4.3　构建三状态翻转器

三状态翻转器就是能够显示图像的 3 个版本的翻转器。除了原始图像和当用户将鼠标移动到图像上时显示的图像之外，图像还有第三个版本，这个版本在按钮本身被单击时显示，见图 4-5。



图 4-5　当单击按钮时，显示第三个图像（在这种灰度图中很难看到这种效果，在本书的配套 Web 站点上可以看到完整的效果）

脚本 4-5（HTML 文件）看起来几乎与前面的脚本 4-3 完全相同。实际上，只是文档的标题和调用的外部 JavaScript 文件名有差异。这正说明了将所有 JavaScript 放进外部文件中是多么有意义。你可以在页面中增加功能，而不需要修改 HTML 页面。

脚本 4-5　由于将 JavaScript 放在外部文件中，三状态翻转器的 HTML 实质上与两状态翻转器相同

```
<!DOCTYPE html>
<html>
<head>
     <title>Three-state Rollovers</title>
     <script src="script03.js"></script>
     <link rel="stylesheet" href= "script01.css">
</head>
<body>
     <a href="next1.html"><img src="images/button1_off.gif" alt="button1" id="button1"></a>  
     <a href="next2.html"><img src= "images/button2_off.gif" alt="button2" id="button2"></a>
</body>
</html>
```

在脚本 4-6（外部 JavaScript 文件）中，与脚本 4-4 相比只有几处修改。我们不必再次逐行研究整个脚本，只需关注修改的地方。我们讨论的脚本部分以**粗体**显示。

脚本 4-6　这个脚本可以产生三状态翻转器

```
window.onload = rolloverInit;

function rolloverInit() {
     for (var i=0; i<document.images.length;i++) {
        if (document.images[i].parentNode.tagName == "A") {
           setupRollover(document.images[i]);
        }
     }
}

function setupRollover(theImage) {
     theImage.outImage = new Image();
     theImage.outImage.src = theImage.src;
     theImage.onmouseout = function() {
        this.src = this.outImage.src;
     }

     theImage.clickImage = new Image();
     theImage.clickImage.src = "images/" + theImage.id + "_click.gif";
```

```
      theImage.onclick = function() {
         this.src = this.clickImage.src;
      }

      theImage.overImage = new Image();
      theImage.overImage.src = "images/" + theImage.id + "_on.gif";
      theImage.onmouseover = function() {
         this.src = this.overImage.src;
      }
   }
```

4

⇛ 构建三状态翻转器

1. `theImage.clickImage = new Image();`
   `theImage.clickImage.src = "images/" + theImage.id + "_click.gif";`

在 setupRollover()函数中，我们需要添加第三个图像属性，它用于单击状态。在第一行中，我们创建一个新的图像对象，它将包含图像的 clickImage 版本。第二行设置 clickImage 的来源。它将"images/"和图像的 id 拼接起来，再加上"_click.gif"，从而在运行时构造出源文件名。

2.
```
theImage.onclick = function() {
   this.src = this.clickImage.src;
}
```

这一行告诉浏览器，当用户在图像上单击鼠标时，应该做什么。在这种情况下，将图像的来源设置为图像的 clickImage 版本。

✔提示

- ❑ 如果你想在自己的网站应用功能类似的脚本，7.7 节的脚本 7-9 给出了较完整的版本，13.8 节的脚本 13-19 是最终版本。

4.4　由链接触发翻转器

在前面的示例中，用户都是通过将鼠标移动到图像上来触发翻转器。但是，也可以让翻转器在用户将鼠标指向文本链接时翻转，如图 4-6 和图 4-7 所示。这个 HTML 页面非常简单，只有一个链接和一个图像，见脚本 4-7。我们将前面示例中使用的脚本修改为脚本 4-8 这样，从而实现这个翻转器。

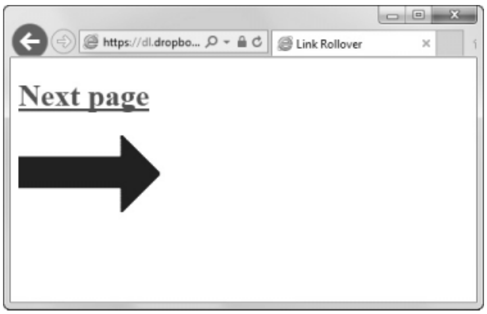


图 4-6　文本链接是这个翻转器的触发元素



图 4-7　当用户将鼠标指向链接时，下面的图像会发生变化

脚本 4-7　这个脚本显示由文本链接触发的翻转器的 HTML

```
<!DOCTYPE html>
<html>
<head>
     <title>Link Rollover</title>
     <script src="script04.js"></script>
     <link rel="stylesheet" href="script01.css">
</head>
<body>
     <h1><a href="next.html" id="arrow">Next page</a></h1>
     <img src="images/arrow_off.gif" id="arrowImg" alt="arrow">
</body>
</html>
```

脚本 4-8　这里是由文本链接触发的翻转器的 JavaScript

```
window.onload = rolloverInit;

function rolloverInit() {
     for (var i=0; i<document.links.length; i++) {
        var linkObj = document.links[i];
        if (linkObj.id) {
           var imgObj = document.getElementById(linkObj.id + "Img");
           if (imgObj) {
              setupRollover(linkObj,imgObj);
           }
        }
     }
}

function setupRollover(theLink,theImage) {
     theLink.imgToChange = theImage;
     theLink.onmouseout = function() {
        this.imgToChange.src = this.outImage.src;
     }
     theLink.onmouseover = function() {
        this.imgToChange.src = this.overImage.src;
     }

     theLink.outImage = new Image();
     theLink.outImage.src = theImage.src;

     theLink.overImage = new Image();
     theLink.overImage.src = "images/" + theLink.id + "_on.gif";
}
```

⇒ 由链接触发翻转器

1. `function rolloverInit() {`
   `for (var i=0; i<document.links.length; i++) {`

首先开始 `rolloverInit()`函数，然后开始一个循环，这与本章前面的示例很相似。但是，以前寻找的是图像（`document.images.length`），而这里要寻找链接（`document.links.length`）。这个循环首先将计数器变量 `i` 设置为 0。在每次迭代中，如果 `i` 的值小于文档中的链接数量，就将 `i` 递增 1。

2. `var linkObj = document.links[i];`

我们创建 `linkObj` 变量并将它设置为当前链接。

3. if (linkObj.id) {
```
    var imgObj = document.getElementById(linkObj.id + "Img");
```
如果 linkObj 有 id，就检查页面上是否有对应的图像元素，这个元素的 id 是 linkObj 的 id 加上 Img。如果有这样的元素，就将它放进新变量 imgObj 中。

4. if (imgObj) {
```
    setupRollover(linkObj,imgObj);
```
如果 imgObj 存在，那么调用 setupRollover()函数，并将链接对象和图像对象传递给它。

5. function setupRollover(theLink, theImage) {
```
    theLink.imgToChange = theImage;
```
在 setupRollover()中，首先利用步骤 4 中传递给它的链接和图像参数，在链接对象中添加一个新的属性 imgToChange。JavaScript 需要知道当鼠标停留在链接上时要改变哪个图像，imgToChange 用来存储这一信息。

6. theLink.onmouseout = function() {
```
    this.imgToChange.src = this.outImage.src;
  }
  theLink.onmouseover = function() {
    this.imgToChange.src = this.overImage.src;
  }
```
mouseOver 和 mouseOut 被触发时，与本章前面的示例稍有不同：现在重新设置 this.imgToChange.src，而不是设置 this.src 本身。

✔提示

- 可以使用这种技术向用户提供预览，让他们提前了解单击所指向的链接之后将看到的内容。例如，假设有一个旅游站点描述苏格兰、塔希提岛和克利夫兰等地的风光。在页面的左边是一栏文本链接，每个旅游目的地有一个链接；在右边是一个预览区域，其中显示一个图像。当用户将鼠标指向一个目的地名称时，在预览区域中显示此地的一张图片。单击链接就会将用户带入一个新页面，这里详细描述此地的著名旅游景点。

4.5　让多个链接触发一个翻转器

到目前为止，你已经看到了在将鼠标移动到某一区域时如何触发翻转器效果。但是，还可以让几个不同的图像触发同一个翻转器。如果需要对多个图像进行说明，这种技术就非常有用。将鼠标移动到其中一个图像上时，就会显示对这个图像的描述。在这个示例中，我们对 Leonardo da Vinci（达芬奇）的三项发明的图像应用这种技术。当用户将鼠标移动到其中一个图像上时，对这个图像的描述就会出现在文本框中。描述本身是另一个图像。更精确地说，它是三个图像，对于三项发明各有一个图像。图 4-8 显示脚本 4-9（HTML）、脚本 4-10（CSS）和脚本 4-11（JavaScript）的效果。与本书中的大多数脚本一样，它是在前面示例的基础上构建的，所以我们只解释新概念。脚本 4-8 与脚本 4-11 相比只有几行代码不一样。

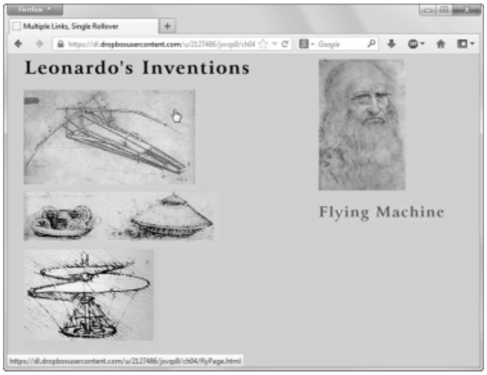


图 4-8　这个页面上有 3 个交互式图像，一个飞行器、一个水箱和一个直升机。当用户将鼠标移动到图像上时，对它的描述就会出现在达芬奇的画像下面

脚本 4-9　注意，这个页面上的链接和图像都有唯一的 id

```
<!DOCTYPE html>
<html>
<head>
     <title>Multiple Links, Single Rollover </title>
     <script src="script05.js"></script>
     <link rel="stylesheet" href="script02.css">
</head>
<body>
     <div id="captionDiv">
        <img src="images/DaVinci.jpg" width="144" height="219" alt="DaVinci">
        <img src="images/bg.gif" id="captionField" alt="Text Field">
     </div>
     <div id="inventionDiv">
        <img src="images/leoText.gif" id="heading" alt="Leonardo's Inventions">
        <a href="flyPage.html" class= "captionField" id="flyer"><img src= "images/flyer.gif" width="293"
        ➝height="165" alt="Flying Machine" id="flyerImg"></a>
        <a href="tankPage.html" class= "captionField" id="tank"><img src="images/tank.gif" width="325"
        ➝height="92" alt="Tank" id="tankImg"></a>
        <a href="heliPage.html" class="captionField" id="helicopter"><img src="images/helicopter.gif"
        ➝width="224" height="160" alt="Helicopter" id="helicopterImg"></a>
     </div>
</body>
</html>
```

脚本 4-10　在这个 CSS 文件中，我们定义在 HTML 中引用的类

```
body {
     background-color: #EC9;
}

img {
     border-width: 0;
}

#captionDiv {
     float: right;
     width: 210px;
     margin: auto 50px;
}
```

```
#captionField {
     margin: 20px auto;
     width: 208px;
     height: 27px;
}

#inventionDiv {
     width: 375px;
     margin-left: 20px;
}

#heading {
     margin-bottom: 20px;
     width: 375px;
     height: 26px;
}
```

4

脚本 4-11 这个脚本演示如何使用多个链接触发一个翻转器

```
window.onload = rolloverInit;

function rolloverInit() {
     for (var i=0; i<document.links.length; i++) {
        var linkObj = document.links[i];
        if (linkObj.className) {
           var imgObj = document.getElementById(linkObj.className);
           if (imgObj) {
              setupRollover(linkObj,imgObj);
           }
        }
     }
}

function setupRollover(theLink,textImage) {
     theLink.imgToChange = textImage;
     theLink.onmouseout = function() {
        this.imgToChange.src = this.outImage.src;
     }
     theLink.onmouseover = function() {
        this.imgToChange.src = this.overImage.src;
     }

     theLink.outImage = new Image();
     theLink.outImage.src = textImage.src;

     theLink.overImage = new Image();
     theLink.overImage.src = "images/" + theLink.id + "Text.gif";
}
```

⇛ 让多个链接触发一个翻转器

1.
```
if (linkObj.className) {
   var imgObj = document.getElementById(linkObj.className);
```

无法使用翻转图像的 id 计算出改变过的图像的 id，这是因为 id 必须是唯一的，而所有翻转图像必须为改变过的图像的目的地提供同样的值。因此，我们要使用 class 属性（因为多个页面元素可以共享同一个 class）。在这一行上，寻找链接对象的 className。

2. `function setupRollover(theLink, textImage) {`
    `theLink.imgToChange = textImage;`

我们将当前链接对象（theLink）和图像对象（我们称之为 textImage）传递给 setupRollover() 函数。注意，在传递这些对象（也可以称为变量）时，分别将它们称为 linkObj 和 imgObj。

脚本其余部分的工作方式与本章前面的示例相同。

4.6　处理多个翻转器

如果希望触发翻转器的图像本身也是一个翻转器，那么应该怎么办？图 4-9 在前一个示例的基础上进行了改进，演示如何添加这个特性。在将鼠标移动到一个自创图像上时，与前面一样，描述图像会出现，但是现在自创图像本身也会切换为另一个带阴影的图像。这会向用户提供视觉反馈，明确地指出鼠标当前指向的元素（鼠标指针本身的指示效果可能不够明显）。脚本 4-12 是 HTML 页面（除了标题和调用的外部 JavaScript 文件名之外，没有变化），脚本 4-13 显示了与前面的示例相比增加的 JavaScript 代码。

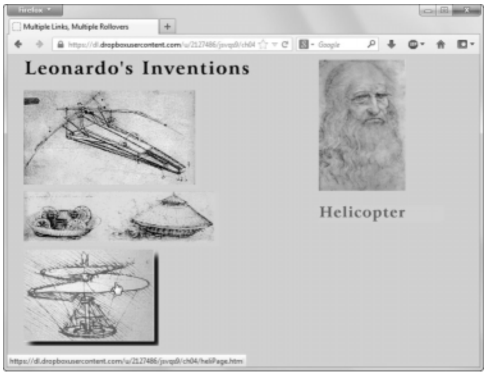


图 4-9　在将鼠标移动到一个图像上时，描述会出现，而且图像本身会加上阴影

脚本 4-12　除了标题和外部脚本的引用之外，这个 HTML 与脚本 4-9 相同

```
<!DOCTYPE html>
<html>
<head>
     <title>Multiple Links, Multiple Rollovers</title>
     <script src="script06.js"></script>
     <link rel="stylesheet" href="script02.css">
</head>
<body>
     <div id="captionDiv">
        <img src="images/DaVinci.jpg" width="144" height="219" alt="DaVinci">
        <img src="images/bg.gif" id="captionField" alt="Text Field">
     </div>
     <div id="inventionDiv">
        <img src="images/leoText.gif" id="heading" alt="Leonardo's Inventions">
        <a href="flyPage.html" class="captionField" id="flyer"><img src="images/flyer.gif" width="293"
        →height="165" alt="Flying Machine" id="flyerImg"></a>
        <a href="tankPage.html" class="captionField" id="tank"><img src="images/tank.gif" width="325"
```

```
        ➝height="92" alt="Tank" id="tankImg"></a>
        <a href="heliPage.html" class="captionField" id="helicopter"><img src="images/helicopter.gif"
        ➝width="224" height="160" alt="Helicopter" id="helicopterImg"></a>
    </div>
</body>
</html>
```

脚本 4-13　这个脚本处理多个翻转器

```
window.onload = rolloverInit;

function rolloverInit() {
     for (var i=0; i<document.links.length; i++) {
        var linkObj = document.links[i];
        if (linkObj.className) {
           var imgObj = document.getElementById(linkObj.className);
           if (imgObj) {
              setupRollover(linkObj,imgObj);
           }
        }
     }
}

function setupRollover(thisLink,textImage) {
     theLink.imgToChange = new Array;
     theLink.outImage = new Array;
     theLink.overImage = new Array;

     theLink.imgToChange[0] = textImage;
     theLink.onmouseout = rollOut;
     theLink.onmouseover = rollOver;

     theLink.outImage[0] = new Image();
     theLink.outImage[0].src = textImage.src;

     theLink.overImage[0] = new Image();
     theLink.overImage[0].src = "images/" + theLink.id + "Text.gif";

     var rolloverObj = document.getElementById(theLink.id + "Img");
     if (rolloverObj) {
        theLink.imgToChange[1] = rolloverObj;

        theLink.outImage[1] = new Image();
        theLink.outImage[1].src = rolloverObj.src;

        theLink.overImage[1] = new Image();
        theLink.overImage[1].src = "images/" + theLink.id + "_on.gif";
     }
}

function rollOver() {
     for (var i=0; i<this.imgToChange.length; i++) {
        this.imgToChange[i].src = this.overImage[i].src;
     }
}

function rollOut() {
     for (var i=0; i<this.imgToChange.length; i++) {
        this.imgToChange[i].src = this.outImage[i].src;
     }
}
```

⇒ 处理多个翻转器

1. theLink.imgToChange = new Array;
```
theLink.outImage = new Array;
theLink.overImage = new Array;
```

添加这些行，是因为这个脚本有更多的图像需要处理（每个翻转器有两个图像）。在每一行中，我们为 theLink 创建一个新属性，每个属性都是一个数组。

2. theLink.imgToChange[0] = textImage;

在前一节中，imgToChange 是一个图像，但是在这里，它是一个包含图像的数组。在这一行中，textImage 被存储在 imgToChange 的第一个元素中。

3. theLink.outImage[0] = new Image();
```
theLink.outImage[0].src = textImage.src;
```

与前面一样，需要存储图像的 outImage 版本，但是这一次它被存储在 outImage 数组的第一个元素中。

4. thisLink.overImage[0] = new Image();
```
thisLink.overImage[0].src = "images/" + theLink.id + "Text.gif";
```

同样，计算出图像的 overImage 版本的文件名并将其存储在 overImage 数组的第一个元素中。

5. var rolloverObj = document.getElementById(theLink.id + "Img");
```
if (rolloverObj) {
```

现在，需要查明这个翻转器是否将触发多个图像，而不只是一个图像。如果是这种情况，HTML 页面上就会有一个对应的元素，它的 id 是当前链接的 id 加上 Img。也就是说，如果正在操作 flyer 链接，就检查页面上是否有 flyerImg 元素。如果有，就将它保存在 rolloverObj 中，并且应该执行下面三步。

6. theLink.imgToChange[1] = rolloverObj;

现在，按照前面设置 imgToChange[0]的方式，将 imgToChange[1]（数组中的第二个元素）设置为新的 rolloverObj。当触发 onmouseout 和 onmouseover 事件处理程序时，两个图像将被替换为它们的替代版本，我们将在稍后看到这一点。

7. theLink.outImage[1] = new Image();
```
theLink.outImage[1].src = rolloverObj.src;
```

这将 outImage 数组的第二个元素设置为图像的 outImage 版本。

8. theLink.overImage[1] = new Image();
```
theLink.overImage[1].src = "images/" + theLink.id + "_on.gif";
```

这里计算出图像的 overImage 版本的文件名并将其存储在 overImage 数组的第二个元素中。

如果由于某种原因，在同一翻转器执行期间还需要改变第三个图像，那么应该对第三个图像对象重复步骤 6 ~ 8。

9. for (var i=0; i<this.imgToChange.length; i++) {
```
  this.imgToChange[i].src = this.overImage[i].src;
}
```

这段代码在 rollOver()函数中，这里是对图像进行切换的地方。因为可能要改变一个或多个图像，所以首先需要了解已经存储了多少个图像，也就是 this.imgToChange.length 的值。在这个示例中，这个值是 2，因为有两个图像需要改变。然后进行两次迭代，将 imgToChange[0]和 imgToChange[1]的来源设置为对应的 overImage 值。

10.
```
for (var i=0; i<this.imgToChange.length; i++) {
   this.imgToChange[i].src = this.outImage[i].src;
}
```

这段代码在 rollOut()函数中，它在本质上与前一步中的代码相同，唯一的差异是将这些图像设置为 outImage 版本的来源值。

4

✔提示

- ❑ 一定要记住，翻转的每个图像必须具有唯一的 id。
- ❑ 如果想让页面上的一些链接触发多个翻转器，而其他链接只是单独的翻转器，那么应该怎么做？没问题，甚至不需要修改 JavaScript。只要步骤 5 中的检查没有在页面上找到替代 id，就不会存储第二个元素，rollOver()和 rollOut()中的循环只使用初始图像。

4.7　创建循环的广告条

在 Web 上冲浪时，常常会见到定期在图像之间切换的广告条。其中一部分是动画 GIF 文件，这种 GIF 文件包含许多帧连续播放的图像，其他的是 Flash 动画。如果希望页面循环显示许多 GIF（无论是否是动画），那么可以使用 JavaScript 来实现，就像脚本 4-15 中那样。这个示例使用 3 个 GIF 并重复地循环显示它们，如图 4-10、图 4-11 和图 4-12 所示。HTML 页面见脚本 4-14。

脚本 4-14　这个 HTML 页面在循环的广告条中加载第一个图像，其他工作由 JavaScript 处理

```
<!DOCTYPE html>
<html>
<head>
     <title>Cycling Banner</title>
     <script src="script07.js"></script>
     <link rel="stylesheet" href="script01.css">
</head>
<body>
     <div class="centered">
        <img src="images/reading1.gif" id="adBanner" alt="Ad Banner">
     </div>
</body>
</html>
```

脚本 4-15　可以使用 JavaScript 在广告条中循环显示图像

```
window.onload = rotate;

var theAd = 0;
var adImages = new Array("images/reading1.gif","images/reading2.gif","images/reading3.gif");

function rotate() {
     theAd++;
     if (theAd == adImages.length) {
```

```
        theAd = 0;
    }
    document.getElementById("adBanner").src = adImages[theAd];

    setTimeout(rotate, 3 * 1000);
}
```



图 4-10　循环广告条显示的第一个图像



图 4-11　循环广告条显示的第二个图像



图 4-12　最后一个图像。加载页面并且广告条开始循环之后，动画会一直运行，不需要用户的干预

⇒ 创建循环广告条

1. `var theAd = 0;`
   `var adImages = new Array("images/reading1.gif","images/reading2.gif","images/reading3.gif");`

脚本首先创建变量 theAd，它在这段代码中被设置了初始值。第二行代码创建一个名为 adImages 的新数组，它包含构成循环广告条的 3 个 GIF 文件的文件名。

2. `function rotate() {`

先建立一个称为 rotate()的新函数。

3. `theAd++;`

这一行将 theAd 的值加 1。

4. `if (theAd == adImages.length) {`
   `    theAd = 0;`

这一行代码检查 theAd 的值是否等于 adImages 数组中成员的数量，如果等于，就将 theAd 的值设置为 0。

5. `document.getElementById("adBanner").src = adImages[theAd];`

页面上循环显示的图像的 `id` 是 `adBanner`。在脚本 4-14 中可以看到，`img` 标签中定义了这个 `id`。这行代码指出，`adBanner` 图像的来源在 `adImages` 数组中，而且 `theAd` 变量的值决定浏览器当前应该使用 3 个 GIF 中的哪一个。

6. `setTimeout(rotate, 3 * 1000);`

这一行告诉脚本每隔多长时间改变广告条中的 GIF。可以使用内置的 JavaScript 命令 `setTimeout` 指定一个操作应该间隔多长时间执行一次（时间的单位是毫秒）。在这个示例中，每 3000 毫秒（即 3 秒）调用一次 `rotate()`函数，所以广告条中的 GIF 每 3 秒改变一次。

✔提示

- 你可能会问，为什么要用 JavaScript 实现循环广告条，创建一个动画 GIF 不就行了吗？原因之一是，这使你能够在广告条中使用 JPEG 或 PNG，因此图像的质量更好。通过使用这些高质量的图像，可以在广告条中显示照片。
- 与本章中前面的示例不同，这里的图像并不预先缓存。每个图像在第一次显示时从服务器下载。这是因为在 `adImages` 数组中可能有任意数量的图像，如果用户在这个页面上停留的时间很短，只会看到其中的两三个图像，那么强迫他们下载大量图像（比如 100 个）是不妥当的。

4.8　在循环广告条中添加链接

广告条常常用来做广告，而且常常希望在广告条中建立链接，让访问者可以通过单击链接进入与广告相关的站点。脚本 4-16 显示的 HTML 页面与前一个示例相比只有一点不同：它在 `img` 标签外边添加了一个链接。脚本 4-17 是前一个脚本的一个变体。在这个脚本中，我们将添加一个新数组。这个新数组包含当用户单击广告条时他们将转到的目的地。在这个示例中，Eat at Joe's 广告条会把用户带到 negrino.com，Drink more Java 链接到 sun.com，Heartburn 链接到 microsoft.com，见图 4-13。

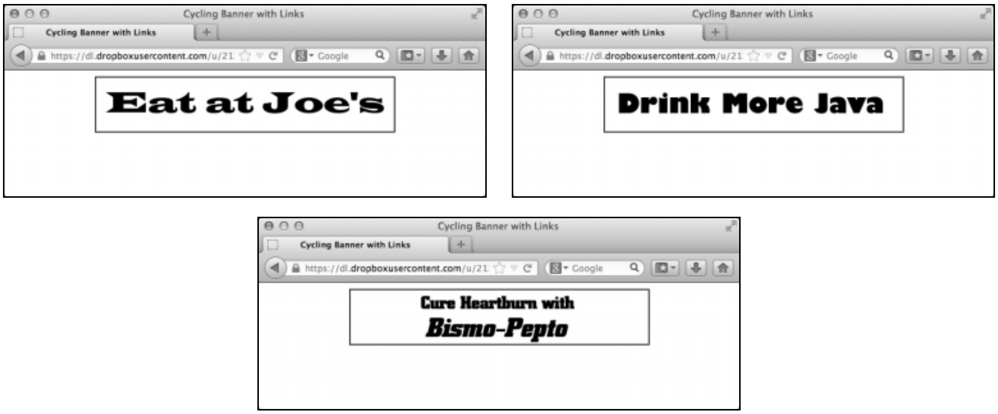


图 4-13　这 3 个图像都是链接，单击它们就会进入三个 Web 站点之一

脚本 4-16　广告条所需的 HTML

```
<!DOCTYPE html>
<html>
<head>
     <title>Cycling Banner with Links</title>
     <script src="script08.js"></script>
     <link rel="stylesheet" href="script01.css">
</head>
<body>
     <div class="centered">
        <a href="linkPage.html"><img src="images/banner1.gif" id="adBanner" alt="ad banner"></a>
     </div>
</body>
</html>
```

脚本 4-17　这个脚本演示如何将循环广告条转换为真正可单击的广告条

```
window.onload = initBannerLink;

var theAd = 0;
var adURL = new Array("negrino.com","sun.com","microsoft.com");
var adImages = new Array("images/banner1.gif","images/banner2.gif","images/banner3.gif");

function initBannerLink() {
     if (document.getElementById("adBanner").parentNode.tagName == "A") {
        document.getElementById("adBanner").parentNode.onclick = newLocation;
     }

     rotate();
}

function newLocation() {
     document.location.href = "http://www." + adURL[theAd];
     return false;
}

function rotate() {
     theAd++;
     if (theAd == adImages.length) {
        theAd = 0;
     }
     document.getElementById("adBanner").src = adImages[theAd];

     setTimeout(rotate, 3 * 1000);
}
```

⇒ 在循环广告条中添加链接

1. `window.onload = initBannerLink;`

当窗口完成加载时，触发 initBannerLink()函数。

2.
```
if (document.getElementById("adBanner").parentNode.tagName == "A") {
    document.getElementById("adBanner").parentNode.onclick = newLocation;
  }
  rotate();
```

这段代码在 initBannerLink()函数中，它首先检查 adBanner 对象是否包围在链接标签中。如果是

这样，那么当单击链接时，将调用 newLocation()函数。最后，调用 rotate()函数。

3. document.location.href = "http://www." + adURL[theAd];
   return false;

这段代码在新函数 newLocation 中，将 document.location.href 对象（换句话说，就是当前文档窗口）设置为文本字符串 http://www.（注意点号）加上 adURL 的值。因为 adURL 是一个数组，所以需要指定数组的一个成员。成员的编号存储在 theAd 中，产生的字符串可以是三个链接之一（取决于用户何时进行单击）。最后，它返回 false，告诉浏览器不应再加载这个 href。如果不这样做，浏览器就会加载这个 URL 两次。我们已经在 JavaScript 中处理了所有工作，所以不需要再加载 href。

✔提示

- ❑ adURL 数组中的成员数量必须与 adImages 数组相同，这样这个脚本才能正确地工作。

4.9　建立循环式幻灯片

Web 站点上的幻灯片每次向用户显示一个图像，并且让用户能够控制显示图像的进度（既可以向前，也可以向后）。JavaScript 向用户提供所需的交互式控制能力。脚本 4-18 显示了必需的 HTML，脚本 4-19 是在页面上添加幻灯片所需的 JavaScript。

脚本 4-18　这个 HTML 页面创建幻灯片

```
<!DOCTYPE html>
<html>
<head>
     <title>Image Slideshow</title>
     <script src="script09.js"></script>
     <link rel="stylesheet" href="script01.css">
</head>
<body>
     <div class="centered">
        <h1>Welcome, Robot Overlords!</h1>
        <img src="images/robot1.jpg" id="myPicture" width="200" height="400" alt="Slideshow">
        <h2><a href="previous.html" id="prevLink">&lt;&lt; Previous</a>  <a href="next.html"
        ➝id="nextLink">Next &gt;&gt;</a></h2>
     </div>
</body>
</html>
```

脚本 4-19　这个脚本构建一个幻灯片，用户可以通过单击链接控制图像的前后切换

```
window.onload = initLinks;

var thePic = 0;
var myPix = new Array("images/robot1.jpg","images/robot2.jpg","images/robot3.jpg");

function initLinks() {
     document.getElementById("prevLink").onclick = processPrevious;
     document.getElementById("nextLink").onclick = processNext;
}

function processPrevious() {
     if (thePic == 0) {
        thePic = myPix.length;
     }
```

```
        thePic--;
        document.getElementById("myPicture").src = myPix[thePic];
        return false;
    }

    function processNext() {
        thePic++;
        if (thePic == myPix.length) {
            thePic = 0;
        }
        document.getElementById("myPicture").src = myPix[thePic];
        return false;
    }
```

这个脚本构建的幻灯片是循环式的（wrap around），也就是说，如果超过了图像列表的末尾，就会返回到开头，反之亦然。图 4-14 显示这个幻灯片的效果。

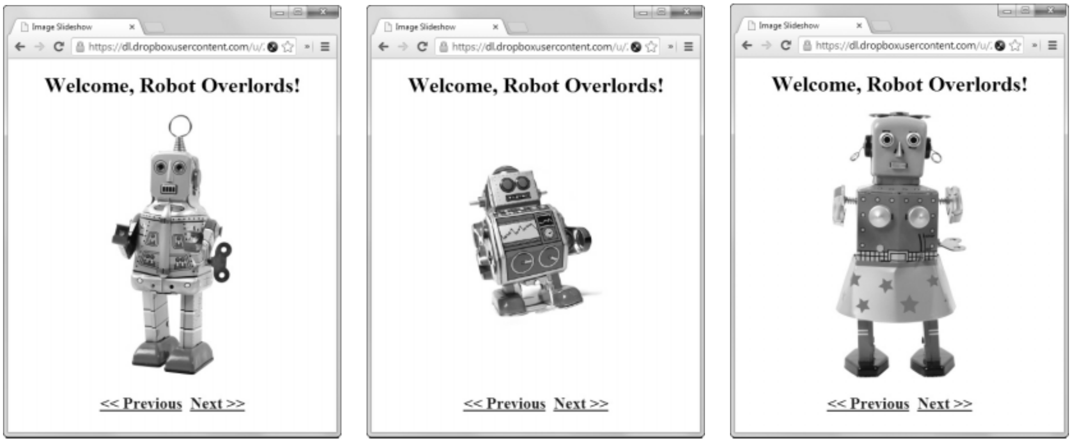


图 4-14　单击 Previous 或 Next 链接，就会分别调用 processPrevious()或 processNext()函数

⇒ 建立循环式幻灯片

1. window.onload = initLinks;

当窗口完成加载时，触发 initLinks()函数。

2. function initLinks() {

```
    document.getElementById("prevLink").onclick = processPrevious;
    document.getElementById("nextLink").onclick = processNext;
  }
```

这个函数为 Previous 或 Next 链接设置 onclick 事件处理程序。

3. function processPrevious() {

```
    if (thePic == 0) {
      thePic = myPix.length;
```

这个函数使幻灯片向前切换图像。它首先检查 thePic 是否等于 0。如果是，这个函数就获取 myPix 数组中的图像数量。

4. `thePic--;`
```
document.getElementById("myPicture").src = myPix[thePic];
```
第一行将 thePic 的值减 1。下一行将 myPicture 的 src 属性设置为 myPix 数组中与 thePic 的当前值对应的元素。

5. `thePic++;`
```
if (thePic == myPix.length) {
  thePic = 0;
}
document.getElementById("myPicture").src = myPix[thePic];
```
这段代码在 processNext()函数中，它使幻灯片向后切换图像，其工作方式与 processPrevious()函数非常相似。它首先将 thePic 的值加 1，然后检查 thePic 的值是否等于 myPix 数组中成员的数量。如果是，就将 thePic 设置为 0。下一行设置 myPicture 的 src 属性。

4.10　显示随机图像

如果你的站点包含大量图形，或者要显示数字化艺术作品，那么可能希望在用户进入站点时从图像集合中随机选择要显示的图像。同样，JavaScript 可以实现这种功能！脚本 4-20 是所需的 HTML，相当简单，脚本 4-21 是 JavaScript。图 4-15 显示脚本的结果，在这个示例中图像是三种毛绒玩具（分别是狮、虎和熊）。

脚本 4-20　这个简单的 HTML 创建显示随机图像的页面

```
<!DOCTYPE html>
<html>
<head>
     <title>Random Image</title>
     <script src="script10.js"></script>
     <link rel="stylesheet" href="script01.css">
</head>
<body>
     <img src="images/spacer.gif" width="305" height="312" id="myPicture" alt="some image">
</body>
</html>
```

脚本 4-21　可以用这个脚本在页面上显示随机图像，它使用 JavaScript 的 Math.random 方法生成一个随机数

```
window.onload = choosePic;

var myPix = new Array("images/lion.jpg","images/tiger.jpg","images/bear.jpg");

function choosePic() {
     var randomNum = Math.floor (Math.random() * myPix.length);
     document.getElementById("myPicture").src = myPix[randomNum];
}
```



图 4-15　根据脚本生成的随机数的值，用户会看到三种毛绒玩具之一

⇒ 显示随机图像

1. var myPix = new Array("images/lion.jpg", "images/tiger.jpg","images/bear.jpg");

与前面一样，建立一个包含三个图像的数组，并且将它赋值给变量 myPix。

2. var randomNum = Math.floor(Math.random() * myPix.length);

变量 randomNum 被设置为一个数学表达式的值。这个表达式最好是从内向外读。Math.random 生成一个 0 ~ 1 的随机数，然后将这个数字乘以 myPix.length，即数组中的成员数量（在这个示例中是 3）。Math.floor 将结果向下取整，这意味着最终产生 0、1 和 2 中的一个数字。

3. document.getElementById("myPicture").src = myPix[randomNum];

这行代码根据 myPix 数组设置 myPicture 图像的来源，当前采用的值由 randomNum 的值决定。

4.11　随机开始循环显示图像

如果有许多图像需要显示，你可能不希望在每次加载页面时都从同样的图像开始显示。脚本 4-22 是 HTML，脚本 4-23 是 JavaScript，它组合了前面用于循环广告条的代码和随机图像的代码。

脚本 4-22　HTML 文件中有一个分隔线 GIF，这是广告条出现之前的占位图像

```
<!DOCTYPE html>
<html>
<head>
     <title>Cycling Random Banner</title>
     <script src="script11.js"></script>
     <link rel="stylesheet" href="script01.css">
</head>
<body>
     <div class="centered">
        <img src="images/spacer.gif" id="adBanner" alt="Ad Banner">
     </div>
</body>
</html>
```

脚本 4-23　这个脚本可以从一个随机图像开始循环显示图像

```
window.onload = choosePic;

var theAd = 0;
```

```
var adImages = new Array("images/reading1.gif","images/reading2.gif","images/reading3.gif");

function choosePic() {
     theAd = Math.floor(Math.random() * adImages.length);
     document.getElementById("adBanner").src = adImages[theAd];

     rotate();
}

function rotate() {
     theAd++;
     if (theAd == adImages.length) {
        theAd = 0;
     }
     document.getElementById("adBanner").src = adImages[theAd];

     setTimeout(rotate, 3 * 1000);
}
```

⇛ 随机开始循环显示图像

1. var adImages = new Array("images/reading1.gif", "images/reading2.gif", "images/reading3.gif");

与前面的示例一样，建立一个含有三个图像的数组并且将数组赋值给一个变量。

2. function rotate() {

这个函数与脚本 4-15 中的 rotate()函数相同，其工作方式的细节请参见 4.7 节中的解释。

4

第 5 章

窗口与框架

在Web 浏览器中，窗口是最重要的接口元素。不出所料，JavaScript 提供了很多用来操纵窗口的工具。

JavaScript 操纵窗口的方式同操纵框架类似。这非常合理，因为框架仅仅是整个浏览器窗口中的另一种文档窗口。

然而在过去的几年中，框架不再流行，甚至从 HTML5 中完全移除（iframe 除外）。因此，本章的框架部分会将重点集中在如何通过 JavaScript 让 iframe 变得更有用。

目前需要了解的 HTML 知识见表 5-1。

表5-1　目前需要了解的HTML知识——框架

标　签	属　性	意　义
`iframe`		内部框架，它显示在进行调用的HTML页面中
	`name`	JavaScript也可以使用这个属性引用iframe
	`src`	iframe页面的URL

5.1　防止页面显示在框架中

别人可以将你的页面加载进他们站点上的框架中，使你的页面看起来就像是他们提供的内容。在 JavaScript 中，窗口形成一个层次结构，父窗口处于这个层次结构的顶层。当别人“拦截”你的页面时，他们会迫使它成为父窗口的子框架。图 5-1 显示当页面作为别人站点的一部分显示时的效果。可以使用脚本防止页面被拦截，并且迫使页面总是单独显示在浏览器窗口中。这需要两个脚本。脚本 5-1 总是应该单独显示的 HTML 页面，其中包含调用 JavaScript 的<script>标签；脚本 5-2 是我们将在下一步描述的 JavaScript 文档。

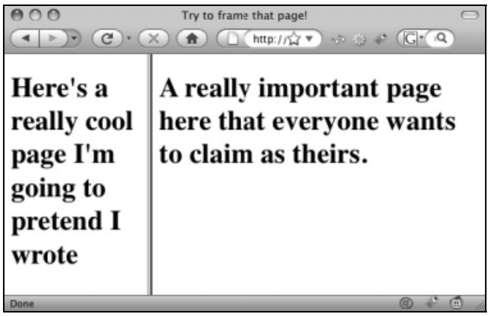


图 5-1　显示在别人框架中的页面

脚本 5-1　这是别人可能想拦截的 HTML 页面

```
<!DOCTYPE html>
<html>
<head>
     <title>Can't be in a frame</title>
     <script src="script01.js"></script>
</head>
<body>
     <h1>A really important page here that everyone wants to claim as theirs.</h1>
</body>
</html>
```

脚本 5-2　JavaScript 可以迫使页面总是单独显示

```
if (top.location != self.location) {
     top.location.replace(self.location);
}
```

⇛ 使页面单独显示

1. `if (top.location != self.location) {`

首先，检查当前页面（`self`）的位置是否处于浏览器窗口层次结构的最顶层。如果是这样，就不需要作任何处理。

2. `top.location.replace(self.location);`

如果当前页面不在顶层，就将顶层页面替换为当前页面的位置。这迫使当前窗口显示我们的页面，而且只显示我们的页面。图 5-2 显示这种效果。



图 5-2　防止恶意拦截框架集之后的页面

✔提示

- ❑ 可以直接将 `top.location` 设置为 `self.location`，但这有一个副作用：用户无法再使用浏览器的 back 按钮。如果他们尝试通过单击 back 按钮返回前一个页面，就会自动跳回当前页面。使用 `replace()`方法会替换浏览器历史中的当前页面，这使 back 按钮能够显示前一个页面。

5.2　设置目标

所谓的 iframe 是一种内联框架。也就是说，它是一种可以嵌入常规 HTML 页面中的框架，并非必须置于框架集之内。与一般的框架一样，iframe 也是一份独立的 HTML 文档。可以将 iframe 作为一段脚本的目标，这样便可以在脚本的控制下，实时创建内容并脱离框架集将其显示在页面中。

在接下来的示例中，我们会看到一份常规的 HTML 页面，其中包含一小块区域，即 iframe。在主内容区域的链接可以通过目标区域作用于 iframe。要使用 HTML 加载 iframe，可以用`<a>`标签的 `target` 属性，这里就是这么做的。

脚本 5-3 的代码可以实现只需点一下链接即可将所选页面载入 iframe。对应的 CSS 是脚本 5-4；载入 iframe 的初始页面是脚本 5-5；请求设置目标的 JavaScript 是脚本 5-6。还有另外三个没有显示的

简单 HTML 页面，也可以加载到 iframe 中。结果如图 5-3 所示。



图 5-3　在主窗口中单击链接，触发 iframe 更新

脚本 5-3　此页面创建 iframe 并调用外部 JavaScript

```
<!DOCTYPE html>
<html>
<head>
     <title>iframe 1</title>
     <script src="script02.js"></script>
     <link rel="stylesheet" href="script01.css">
</head>
<body>
     <iframe src="iframe01.html" id="icontent" name="icontent"></iframe>
     <h1>Main Content Area</h1>
     <h2>
     <a href="page1.html">Link 1</a><br>
     <a href="page2.html">Link 2</a><br>
     <a href="page3.html">Link 3</a>
     </h2>
</body>
</html>
```

脚本 5-4　利用 CSS 样式化主页面并为 iframe 设定位置

```
body {
     background-color: #FFF;
}

iframe#icontent {
     float: right;
     border: 1px solid black;
     width: 350px;
     height: 300px;
     margin-top: 100px;
}
```

脚本 5-5　显示在 iframe 中的初始页面

```
<!DOCTYPE html>
<html>
<head>
     <title>Content iframe</title>
</head>
<body>
```

```
    Please load a page
</body>
</html>
```

脚本 5-6 需要用 JavaScript 来设置框架作为目标有很多原因

```
window.onload = initLinks;

function initLinks() {
    for (var i=0; i<document.links.length; i++) {
       document.links[i].target = "icontent";
    }
}
```

⇛ 设置框架目标

1. `window.onload = initLinks;`

在加载页面时，调用 `initLinks()`函数。

2.
```
for (var i=0; i<document.links.length; i++) {
    document.links[i].target = "icontent";
  }
```

`initLinks()`函数循环遍历页面上的所有链接。当发现链接时，它将 `target` 属性设置为字符串 `"icontent"`。这就需要用到的所有代码。

✔提示

- 如果访问者关闭了 JavaScript 功能，那么单击第一个链接时，页面会加载进主窗口，而不是 `icontent iframe`。很遗憾，但这就是框架的工作方式。
- 没有必要通过脚本设置目标，但通过编程方式设置目标仍旧比较方便。通常，编写主内容区域的开发人员不清楚 iframe 开发人员想要什么，反之亦然。在硬编码方式下，需要将 `target` 写到充斥整个网站的无数的<a>标签中。相比之下，将 `target` 写入一行 JavaScript 代码中显然更灵活。
- 在脚本 5-3 中，通常只设置 iframe 的 `id` 就可以了。不过，有些浏览器要求使用 `name` 属性（如 Firefox 和 Internet Explorer），因此，这里对两者都进行了设置。

5.3 用 JavaScript 加载 iframe

当然，不仅使用 JavaScript 设置目标，你可能希望做更多事情，比如加载其他 HTML 页面。本节讲解应该怎么做。同样，我们有一个建立 iframe 的主页面，这个页面实质上与脚本 5-13 相同。还有一个提供 iframe 初始内容的页面，它与脚本 5-5 相似。所需的 JavaScript 如脚本 5-7 所示。

脚本 5-7 这个脚本把 HTML 页面加载进 `icontent` iframe

```
window.onload = initLinks;

function initLinks() {
    for (var i=0; i<document.links.length; i++) {
       document.links[i].onclick = setContent;
    }
}

function setContent() {
```

```
    document.getElementById("icontent").contentWindow.document.location.href = this.href;
    return false;
}
```

⇒ 用 JavaScript 加载 iframe

1. for (var i=0; i<document.links.length; i++) {
    document.links[i].onclick = setContent;
  }

在脚本 5-6 的作用下，initLinksys()函数会在页面加载时被调用。这样一来，我们对页面中所有的链接都进行了设置，当其被单击时，就会调用 setContent()函数。

2. function setContent() {
    document.getElementById("icontent").contentWindow.document.location.href = this.href;

在这个示例中，单击链接就会触发 setContent()函数，这个函数将新页面加载进 iframe，结果见图 5-4。这是通过找到页面中的 icontent 元素，并将它的 contentWindow.document.location.href 重置为 thishref 实现的。也就是说，我们找到给定 id 的元素（在这里就是 icontent）并获取元素的 contentWindow，然后获取它包含的 document。之后，再获取 document 的 location 并重置 href，而这个 href 就是用户选择加载的 URL。



图 5-4 当单击主窗口中的链接时，icontent iframe 中会加载新页面

3. return false;

最后，setContent()返回 false，意味着告知浏览器不要将 href 一并载入主窗口。否则两者都会被载入。这是因为所有的功能都通过 JavaScript 实现了，所以无需载入 href。

5.4 iframe 的使用

由于 JavaScript 可以实时创建页面内容，因此在基于用户的选择来加载 iframe 页面方面是非常有用的。脚本 5-8 是一个常规的 HTML 页面，其中加载了之前常用的虚拟 iframe 页面。脚本 5-9 详见下文，创建了一个页面并加载到了名为 icontent 的 iframe 中。运行结果如图 5-5 所示。

脚本 5-8 这个页面中有一个位于右侧的 iframe，还包括一些链接说明

```
<!DOCTYPE html>
<html>
```

```
<head>
     <title>iframe 3</title>
     <script src="script04.js"></script>
     <link rel="stylesheet" href="script01.css">
</head>
<body>
     <iframe src="iframe01.html" id="icontent" name="icontent"></iframe>
     <h1>Main Content Area</h1>
     <h2>
     <a href="#">Link 1</a><br>
     <a href="#">Link 2</a><br>
     <a href="#">Link 3</a>
     </h2>
</body>
</html>
```


脚本 5-9 将内容添加至名为 icontent 的 iframe 中的脚本

```
window.onload = initLinks;

function initLinks() {
     for (var i=0; i<document.links.length; i++) {
         document.links[i].onclick = writeContent;
         document.links[i].thisPage = i+1;
     }
}

function writeContent() {
     var newText = "<h1>You are now looking at example " + this.thisPage + ".<\/h1>";

     document.getElementById("icontent").contentWindow.document.body.innerHTML = newText;
     return false;
}
```



图 5-5 这是脚本 5-9 的运行结果，用 JavaScript 写的 iframe

⇛ 如何为 iframe 创建内容

1. `for (var i=0; i<document.links.length; i++) {`
```
    document.links[i].onclick = writeContent;
    document.links[i].thisPage = i+1;
  }
```

`initLinks()`函数首先循环遍历页面上所有的链接。然后对于每个链接设置了两个元素：`onclick` 处理函数和 `thisPage` 属性。后者存有单击链接后会显示的页码，例如链接 0 就是“page 1”，链接 1 就是

“page 2”，以此类推。循环中的 onclick 处理函数可以让每个链接在单击后调用 writeContent()函数。

2.
```
var newText = "<h1>You are now looking at example " + this.thisPage + ".<\/h1>";

document.getElementById("icontent").contentWindow.document.body.innerHTML = newText;
```

这是 writeContent()函数的内容：它首先声明并设置一个变量 newText，然后存入一些字串。考虑到 icontent 元素(5.3 节讲过相关内容)，将它的 contentWindow.document.body.innerHTML 重置为 newText。

✔提示

- 为什么第 2 步中斜杠（“/”）前面会有反斜杠（“\”）？根据标准，浏览器可能会把结束标签的起始字符（“</”）理解为代码行的结尾。反斜杠将斜杠转义，让我们在不引起错误的情况下写出 HTML。

5.5　创建动态 iframe

基于上面的示例，我们可以使用 JavaScript 为 iframe 创建动态内容。创建的内容如脚本 5-10 所示。当用户单击任意一个链接时，JavaScript 向 iframe 写入新的代码。在这个示例中，显示的是页面的名称以及当前会话中用户访问页面的次数。

脚本 5-10　这段脚本会计算 iframe 的内容，并将其写入窗口

```
window.onload = initLinks;
var pageCount = new Array(0,0,0,0);

function initLinks() {
     for (var i=0; i<document.links.length; i++) {
       document.links[i].onclick = writeContent;
       document.links[i].thisPage = i+1;
     }
}

function writeContent() {
     pageCount[this.thisPage]++;

     var newText = "<h1>You are now looking at example " + this.thisPage;
     newText += ".<br>You have been to this page ";
     newText += pageCount[this.thisPage] + " times.<\/h1>";

     document.getElementById("icontent").contentWindow.document.body.innerHTML = newText;
     return false;
}
```

⇒ 加载动态 iframe

1.
```
var pageCount = new Array(0,0,0,0);
```

为了显示加载页面的次数，我们需要一种记录该信息的方法。在这里我们将会使用 pageCount。

2.
```
pageCount[this.thisPage]++;
```

这行代码自增 pageCount 数组，这样我们就可以跟踪到访问特定页面的次数了。

3.
```
var newText = "<h1>You are now looking at example " + this.thisPage;
newText += ".<br>You have been to this page ";
newText += pageCount[this.thisPage] + " times.<\/h1>";
```

这些行实时创建 iframe 的内容。

```
4. document.getElementById("icontent").contentWindow.document.body.innerHTML = newText;
   return false;
```

跟先前的示例一样，我们获取 `icontent` 元素，重置其 `body` 的 `innerHTML` 属性。用两行文字替换 `innerHTML`，结果如图 5-6 所示。由于所有工作都已完成，因此最后以 `return false` 结尾，确保浏览器不会执行其他误操作。



图 5-6　每次单击主窗口中的链接，iframe 的内容就会更新

5.6　在文档之间共享函数

只要主窗口和 iframe 都来自同样的域，那么它们共享单个外部 JavaScript 文件是非常方便的。这里，我们将通过 iframe 加载外部 JavaScript 文件来演示它如何被主窗口调用，如图 5-7 所示。脚本 5-11 是主 HTML 页面。脚本 5-12 是载入到 iframe 中的页面。脚本 5-13 是我们的 JavaScript 文件。

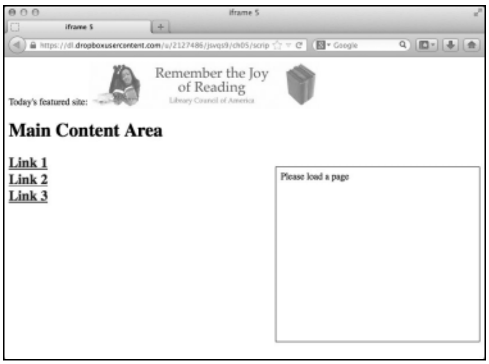


图 5-7　通过调用 iframe 中的代码，更新主页面上的图片

脚本 5-11　页面包含一个 `image` 标签，但不显示任何内容

```
<!DOCTYPE html>
<html>
<head>
```

```
        <title>iframe 5</title>
        <link rel="stylesheet"href="script01.
css">
    </head>
    <body>
        Today's featured site:
        <img src="images/spacer.gif" width="400" height="75" id="adBanner" alt="banner">
        <iframe src="iframe02.html" id="icontent" name="icontent" ></iframe>
        <h1>Main Content Area</h1>
        <h2>
        <a href="#">Link 1</a><br>
        <a href="#">Link 2</a><br>
        <a href="#">Link 3</a>
        </h2>
    </body>
    </html>
```

脚本 5-12　此页面只加载指向其父页面的外部 JavaScript 文件

```
<!DOCTYPE html>
<html>
<head>
     <title>Content iframe</title>
     <script src="script06.js"></script>
</head>
<body>
     Please load a page
</body>
</html>
```

脚本 5-13　这段脚本会更新父页面

```
window.onload = initLinks;
var bannerArray = new Array("images/reading1.gif", "images/reading2.gif", "images/reading3.gif");

function initLinks() {
     for (var i=0; i<parent.document.links.length; i++) {
         parent.document.links[i].onclick = setBanner;
     }

     setBanner();
}

function setBanner() {
     var randomNum = Math.floor(Math.random() * bannerArray.length);

     parent.document.getElementById("adBanner").src = bannerArray[randomNum];
     return false;
}
```

⇒ 在另一个文档中使用函数

1. var bannerArray = new Array("images/reading1.gif","images/reading2.gif","images/reading3.gif");

首先创建一个新数组，其中包含所有可能显示的广告条图像的文件名，并将这个数组赋值给 bannerArray 变量。

2. for (var i=0; i<parent.document.links.length; i++) {
   parent.document.links[i].onclick = setBanner;

现在开始 initLink()函数中的代码。因为要从 iframe 的上下文中调用这个函数，所以设置主窗口中链接的方式与前面的示例有点儿差异。这一次，我们重新设置每个链接的父文档的 onclick 处理程序。

3. setBanner();

作为最后的初始化步骤，调用 setBanner()函数。

4. var randomNum = Math.floor(Math.random() * bannerArray.length);

Set Banner()函数首先计算一个随机数。这一行调用 Math.random()函数，将结果乘以 bannerArray 数组中的成员数量，最后对结果向下取整，从而得到一个 0 到数组成员数量减 1 之间的随机整数。然后，将结果放进 randomNum 变量。

5. parent.document.getElementById("adBanner").src = bannerArray[randomNum];

主窗口仅通过其 id 就可以指向一个 iframe——它的子文档。但在 iframe 指向主窗口之前，它需要显式地指向其 parent。这里，我们获取该元素（一个 window）、其中的 document，以及 adBanner 元素。然后，将 adBanner 的 src 属性设置为数组的当前元素，也就是新的图片名称，将来会显示在页面上。最后，主窗口中的广告条会被设置为随机从该数组中选取。

5

5.7　打开新窗口

开发人员可能希望在不干扰用户正在访问的页面的前提下，通过打开一个新窗口来为用户显示一些附加的信息。例如，我们可能希望为一篇技术论文或新闻报道打开注释和评论。虽然使用 HTML 可以打开新浏览器窗口，不过使用 JavaScript 的话，对新窗口的内容和功能可以有更高的可控性。图 5-8 显示的是一个标准的浏览器窗口，其中所有的部分都已标明。我们创建的窗口可以包括其中的任意部分或者全部。脚本 5-14 是 HTML。脚本 5-15 显示的 JavaScript 文件在图 5-9 所示的页面中创建了一个窗口，单击链接会触发创建新窗口（在本例中，此窗口会显示一幅猫的图片）。



图 5-8　浏览器窗口的各种元素。图中的名称对应 open()命令中可以使用的参数

脚本 5-14　此 HTML 页面调用可以打开新窗口的外部 JavaScript 文件

```
<!DOCTYPE html>
<html>
<head>
     <title>Opening a Window</title>
     <script src="script07.js"></script>
</head>
<body>
     <h1>The Master of the House</h1>
```

```
    <h2>Click on His name to behold He Who Must Be Adored</h2>
    <h2><a href="#" class="newWin">Pixel </a></h2>
</body>
</html>
```



图 5-9　打开新窗口

脚本 5-15　通过下面的脚本打开新窗口

```
window.onload = newWinLinks;

function newWinLinks() {
    for (var i=0; i<document.links.length; i++) {
      if (document.links[i].className == "newWin") {
         document.links[i].onclick = newWindow;
      }
    }
}

function newWindow() {
   var catWindow = window.open("images/pixel1.jpg","catWin","resizable=no,width=350,height=260");
   return false;
}
```

你们可能注意到了，在脚本 5-14 中没有任何 JavaScript 代码，只有一行调用外部 JavaScript 文件的代码。此外，我们在页面中还引入了 link 标签的属性：newWin 类。同 iframe 的示例类似，脚本 5-15 包含一个用于调用函数的 onload 事件处理程序，在这里名为 newWinLinks。这个函数遍历页面中的所有链接，查看是否包含 newWin 类。如果包含 newWin 类，那么单击对应的链接就会调用 newWindow 函数。

⇛ 打开新窗口

1. document.links[i].onclick = newWindow

在 newWinLinks()函数中，我们通过 JavaScript 添加了 newWindow()函数调用作为 onclick 处理程序。

2. function newWindow() {

首先定义一个名为 newWindow()的函数。

3. var catWindow = window.open("images/pixel1.jpg","catWin","resizable=no,width=350,height=260");

catWindow 包含一个新的窗口对象，指向图片文件 pixel1.jpg。新窗口的名称为 catWin。名称是必

填项，因为随后的代码中我们可能需要通过链接或其他脚本来指向此窗口。新窗口的尺寸为 350 像素宽、260 像素高。这些参数是可选的。

✔**提示**

- 在第 3 步中，在 width 和 height 参数的逗号之间不能包含任何字符。如果包含其他字符，很可能在有的浏览器中脚本就无法正常运行了。一般情况下，遇到脚本错误时，需要对脚本进行调试，查找类似的小错误。语法错误是一个主要的报错原因，特别是对新程序员来说。
- Internet Explorer 6 及其更高版本在安全方面制定了一些稀奇古怪、前后不一的策略（是否支持脚本控制窗口等）。如果关闭了安全策略，那么本章的所有示例都能运行，但我们不建议开发人员关闭安全策略或者要求网站的访问者关闭自己的安全策略。此外，在 IE7 及更高版本中，根据用户选项卡式的浏览设置，新窗口可以在新标签页中显示。

在窗口中添加参数

要将图 5-7 中列出的一个或多个参数添加到你的窗口中，可以将其放置在 open()命令中并用引号封装。在希望保留的功能名称后面加上=yes，在不想保留的功能名称后面加上=no（因为=no 通常是默认的，所以开发人员可以跳过这些功能，不予设置）。例如，如果我们想要一个特定尺寸且带有工具栏、地址栏和滚动条的窗口，将下面的代码作为 open()命令的参数即可。"toolbar=yes, location=yes, scrollbars=yes, width=300, height=300"。注意，这样创建的窗口不带有菜单栏或状态栏，大小也不可以改变。

直接忽略某参数其实就相当于将其置为=no（通常是这样的，下面有一些例外情况）。开发人员也可以直接使用参数名称（不带=yes）来开启对应功能。由于存在一些例外情况，所以我们建议显式地声明其打开和关闭状态。如"location=yes,scrollbars=yes"。

图 5-10 从左到右分别显示的是脚本 5-15 在 Windows 中的 Firefox、Windows 中的 IE、Mac 中的 Chrome 和 Mac 中的 Safari 中的显示效果。可以看到，每个浏览器的显示效果都不尽相同。事实上，真正达到我们预期效果的是 Safari。大家自己得到的结果可能跟我们的不一样。拿 Firefox 来说，它将最终控制权交给用户——如果用户选择始终显示状态栏，那么不管脚本是如何编写的，状态栏都会一直显示。

最后，开发人员仍旧需要考虑其用户可能使用的浏览器，并在所有这些浏览器中测试自己的脚本。这很可能意味着测试的时候需要用到 Windows 和 Mac（可能还有 Linux）设备。测试脚本（如果有必要的话，还需要修订脚本）可以让我们在各种浏览器下都能实现预期的效果。



图 5-10　不同的浏览器有不同的窗口默认设置，所以像地址栏之类的元素会始终出现在一些浏览器中，不管脚本中是否声明显示

5.8　为窗口加载不同的内容

在前面的示例中，单击链接会在新窗口中打开一幅图片。但如果页面上有多个链接，而且我们希望这些链接都指向同一个窗口该怎么办？脚本 5-16 演示了这种技术。图 5-11 中的主窗口有 3 个链接。单击其中任意一个都能打开新窗口，然后显示相应的小猫图片。如果切换回主窗口，单击另一个链接的话，小窗口中的图片会被新图片替换掉。

脚本 5-16　通过此脚本，可以打开一个新窗口，其中根据所单击链接的不同，显示各式各样的内容

```
window.onload = newWinLinks;

function newWinLinks() {
     for (var i=0; i<document.links.length; i++) {
        if (document.links[i].className == "newWin") {
           document.links[i].onclick = newWindow;
        }
     }
}

function newWindow() {
     var catWindow = window.open(this.href,"catWin","width=350,height=260");
     catWindow.focus();
     return false;
}
```



图 5-11　单击 3 个链接中的任意一个，都会打开一个小窗口，并用相应的小猫图片填充该窗口

⇛ 为窗口加载不同内容

1. `document.links[i].onclick = newWindow;`

在 `newWinLinks()`函数中，我们通过 JavaScript 添加 `newWindow()`函数调用，作为 `onclick` 处理程序。当 `newWindow()`被调用时，使用 `this.href`，即 HTML 中 `href` 属性的值。

2. `function newWindow() {`

这里我们定义个新的函数：`newWindow()`。

3. `var catWindow = window.open(this.href,"catWin","width=350,height=260");`

这是 `catWindow` 变量,我们正在打开一个新的窗口对象,带有窗口的各项参数。首先传入 `this.href` 的值。新窗口的名称为 `catWin`，`width` 和 `height` 参数设置窗口的尺寸。

4. `catWindow.focus();`

这里使用 `focus()`方法使得我们新打开的窗口位于最前面。我们需要显示某个窗口时，就可以使用 `focus()`。如果同时有多个窗口打开，那么使用 `focus()`可以将指定窗口排列在最顶端。

5. `return false;`

函数以 `return false` 结束，告知 HTML 不要载入 href。

✔提示

- 在第 4 步中使用到了 `blur()`，它是与 `focus()`作用相反的方法。使用 `blur()`方法可以将窗口置于所有打开窗口的最后面。窗口对象的 `focus()`和 `blur()`方法对应于 `onfocus` 和 `onblur` 事件处理程序，通过它们可以在窗口获得或者失去焦点时执行操作。

5

不要阻止弹出窗口！

本章最后两个任务都是关于如何使用 JavaScript 创建和操作窗口的。这些特定的窗口称为弹出窗口，而且被很多网友视为眼中钉。这里我们演示了弹出窗口的一些正面用途，如果你的浏览器不能正常运行这些示例，那可能是在浏览器中关闭了弹出窗口，也可能是正在运行的其他软件禁用了弹出窗口。大部分浏览器可以打开用户指定的弹出窗口，而有些浏览器不行。所以，在大家学习本章内容的时候，务必确保禁用了那些关闭弹出窗口的功能。

然而，有些浏览器（说你呢，Internet Explorer）认为自己比用户聪明，号称为安全起见而强制关闭了脚本中的弹出窗口功能，即便是用户请求打开也不行。

第 6 章

表单处理

如果需要从 Web 站点的用户那里收集信息，那么就需要使用表单。

表单可以包含大多数常见的图形界面元素，包括输入字段、单选按钮、复选框、弹出菜单和输入列表。另外，HTML 表单可以包含密码字段，这种控件可以避免用户的输入被别人偷看。

在填写表单之后，单击表单上的 Submit 按钮，会将表单的信息发送到 Web 服务器，在服务器上服务器端脚本会解释并操作这些数据。通常，将数据存储在数据库中供以后使用。在服务器端存储数据之前，需要确保用户输入的数据是“干净”的，也就是说，数据是准确的且具有正确的格式。JavaScript 是检查数据的好方法，这种技术称为*表单验证*（form validation）。尽管服务器端脚本可以完成验证（而且应该作为预防措施，因为有些用户会在浏览器中关闭 JavaScript 功能），但是在客户机上用 JavaScript 进行验证要快得多，而且用户操作的效率也更高。

在本章中，你将学习如何使用 JavaScript 确保表单包含有效的信息，针对另一个字段中的数据检查一个字段中的数据，以及突出显示错误的信息，让用户知道需要修改什么。

目前需要了解的 HTML 知识见表 6-1。

表6-1　目前需要了解的HTML知识——表单

标　　签	属　　性	意　　义
form		这个标签包含下面的任何标签，构成有效的HTML表单
	action	在Web服务器上处理数据的服务器端脚本的名称
input		这个标签显示不同类型的表单字段，具体取决于type属性的值
	name	主要用来对单选按钮进行分组
	maxlength	用户可以在这个字段中输入的数据的最大长度
	size	在页面上显示的字符数量
	type	所需的输入控件类型，有效值是button、checkbox、image、password、radio、reset、submit和text
	value	预先为这个表单字段设置的值
label		用来为没有内置标签的控件指定标签，比如文本字段、复选框、单选按钮和菜单
	for	将标签与特定元素的id关联起来
option		在select标签中可用的选项
	selected	指出这个选项是否作为默认选项
	value	每个选项的预设值

（续）

标　签	属　性	意　义
select		这种表单字段显示弹出菜单或滚动列表（取决于size属性）
	size	在页面上显示的选项数量。如果这个属性设置为1，或者没有提供这个属性，就会显示弹出菜单

6.1　选择并转移导航菜单

在 Web 上你可能见过许多标准的导航菜单，可以选择菜单中的一个选项并单击 Go 按钮，这样就会转到对应的目的地。例如，许多在线商店使用这样的菜单将用户转到不同的部分。但是，只需用上一点儿 JavaScript，就可以让用户只通过菜单选择进入不同的目的地，而不再需要 Go 按钮（见图 6-1）。这会使用户觉得站点更干净利落而且响应性更好，所以是一种有益的做法。我们将这些 JavaScript 增强菜单称为“选择并转移”菜单，很容易创建它们。HTML 见脚本 6-1，CSS 见脚本 6-2，JavaScript 见脚本 6-3。你不再需要 Go 按钮了！

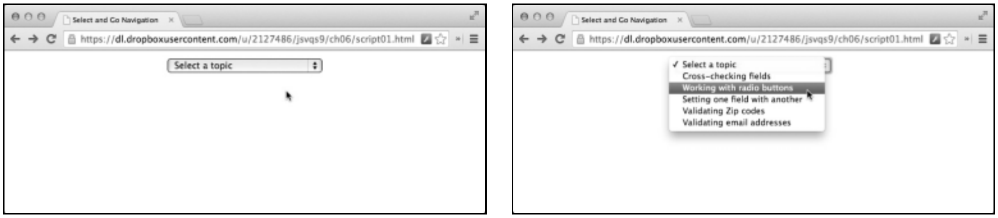


图 6-1　选择这个菜单中的任何选项，就会直接跳到对应的页面，而不需要单独的 Go 按钮

脚本 6-1　选择并转移菜单的 HTML 非常简单

```
<!DOCTYPE html>
<html>
<head>
     <title>Select and Go Navigation</title>
     <link rel="stylesheet" href="script01.css">
     <script src="script01.js"></script>
</head>
<body>
<form action="gotoLocation.cgi" class="centered">
     <select id="newLocation">
        <option selected>Select a topic</option>
        <option value="script06.html">Cross-checking fields</option>
        <option value="script07.html">Working with radio buttons</option>
        <option value="script08.html">Setting one field with another</option>
        <option value="script09.html">Validating Zip codes</option>
        <option value="script10.html">Validating email addresses</option>
     </select>
     <noscript>
        <input type="submit" value="Go There!">
     </noscript>
</form>
</body>
</html>
```

脚本 6-2　这个 CSS 文件的内容并不多，但是当你想添加更多样式时，可以在这里添加

```
.centered {
     text-align: center;
}
```

脚本 6-3　可以使用 JavaScript 和表单实现站点导航

```
window.onload = initForm;
window.onunload = function() {};

function initForm() {
     document.getElementById("newLocation").selectedIndex = 0;
     document.getElementById("newLocation").onchange = jumpPage;
}

function jumpPage() {
     var newLoc = document.getElementById("newLocation");
     var newPage = newLoc.options[newLoc.selectedIndex].value;

     if (newPage != "") {
        window.location = newPage;
     }
}
```

⇒ 创建选择并转移菜单

1. `window.onload = initForm;`
   `window.onunload = function() {};`

在窗口加载时，调用 `initForm()`函数。下一行需要解释一下，因为它是处理某些浏览器的古怪行为的变通方法。

当窗口卸载时（即关闭窗口或者浏览器转到另一个网址），我们调用一个*匿名函数*（anonymous function），即没有名称的函数。在这个示例中，这个函数不但没有名称，而且根本不做任何事情。提供这个函数是因为必须将 `onunload` 设置为*某种东西*，否则，当单击浏览器的 back 按钮时，就不会触发 `onload` 事件，因为在某些浏览器（比如 Firefox 和 Safari）中页面会被缓存。让 `onunload` 执行任何操作，就会使页面不被缓存，因此当用户后退时，会发生 `onload` 事件。

2. `document.getElementById("newLocation").selectedIndex = 0;`
   `document.getElementById("newLocation").onchange = jumpPage;`

在 `initForm()`函数中，第一行获得 HTML 页面上的菜单（它的 `id` 为 `newLocation`），并且将它的 `selectedIndex` 属性设置为零，这会使它显示 Select a topic。

第二行让脚本在菜单选择发生改变时，调用 `jumpPage()`函数。

3. `var newLoc = document.getElementById("newLocation");`

在 `jumpPage()`函数中，`newLoc` 变量查找访问者在菜单中选择的值。

4. `var newPage = newLoc.options[newLoc.selectedIndex].value;`

从方括号中的代码开始，向外依次分析。`newLoc.selectedIndex` 是一个 0 ~ 5 的数字（因为有 6 个菜单选项。记住 JavaScript 的编号常常是基于零的）。得到这个数字之后，接下来获得对应的菜单项的值，这就是我们希望跳转到的网页的名称。然后，将结果赋值给变量 `newPage`。

```
5. if (newPage != "") {
    window.location = newPage;
```

这个条件语句首先检查 newPage 是否非空。换句话说，如果 newPage 有一个值，那么让窗口转到选择的菜单项所指定的 URL。

✔提示

- 这个脚本最棒的一点是，在添加了 JavaScript 函数之后，当添加、修改或改变下拉菜单项时，根本不需要修改这个函数，只需要设置菜单项的值（即菜单项跳转到的 URL）。因此，这个脚本很适合 WYSIWYG 页面编辑器。
- 正如前面提到的，Firefox 会对页面进行缓存，因此在单击 back 按钮时不会触发 onload 事件。前面提供了一种变通方法，另一种可以采用的方法是添加下面这行代码：

  ```
  window.onpageshow = initForm;
  ```

  我们没有采用这种方法，因为它在 Safari（另一种对页面进行缓存的浏览器）中不起作用。但是，如果你专门针对 Firefox 设计网站，那么应该知道有两个非标准的窗口事件处理程序：onpageshow 和 onpagehide，可以使用它们处理只希望在 Firefox 中触发的事件。
- 我们将这些菜单称为"选择并转移"菜单，这个名称并不太好听，但是清楚地表明了它们的作用。你可能会看到对同样的功能有别的称呼。例如，Dreamweaver 将它们称为"跳转菜单"（jump menu）。顺便说一句，如果你是 Dreamweaver 用户，需要一本深入介绍 Dreamweaver 的好书，那么推荐我们所写的 *Dreamweaver: Visual QuickStart Guide*（Peachpit Press）。

照顾不支持 JavaScript 的用户

这个示例的目标是，在使用表单跳转到另一个页面时，利用 JavaScript 避免使用 Go 按钮。但是，如果用户使用老式的不支持 JavaScript 的浏览器，或者关闭了浏览器的 JavaScript 功能，那么该怎么办？没问题，脚本 6-1 会让 Go 按钮只在不支持 JavaScript 时显示，从而照顾这些用户。

用表单从一个页面跳转到另一个页面而不使用 JavaScript 的唯一方法是使用 CGI，CGI 是在 Web 服务器上运行的一个程序。脚本 6-1 在下面这一行中调用 CGI：

```
<form action="gotoLocation.cgi">
```

这个 form 标签有 action 属性，它会调用 CGI。但是，表单 action 属性要求用户单击 submit 按钮，而图 6-1 中没有这样的按钮。但是，在图 6-2 中有 submit 按钮，这是在关闭 JavaScript 功能时显示的页面。下面这些代码行提供按钮（包围在 noscript 标签中），只有在缺少 JavaScript 支持时这些代码才会执行。

```
<noscript>
        <input type="submit" value="Go There!">
</noscript>
```

最酷的是，服务器端脚本只在缺少 JavaScript 支持时被调用。如果用户使用支持 JavaScript 的浏览器，那么 submit 按钮就不会出现，而服务器端脚本就是不必要的。

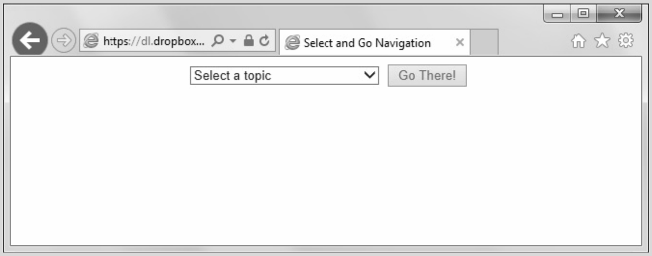


图 6-2　如果用户的浏览器不支持 JavaScript，他们仍然能够在你的站点上跳转，因为 Go There！按钮会自动出现

6.2　动态地改变菜单

常常需要通过弹出菜单向用户提供选择输入的机会，而且希望根据用户在另一个弹出菜单中所做的选择，改变一个或多个弹出菜单的内容。在 Web 站点上，你可能见过这样的情况：站点让你从一个弹出菜单中选择你所在的国家，然后根据你的选择，用州或省的名称填充第二个菜单。在脚本 6-4（HTML）和脚本 6-5（JavaScript）中，我们要使用两个弹出菜单（见图 6-3）。第一个菜单用来选择月份。当用户选择一个月份时，脚本根据所选月份的天数，填充第二个弹出菜单（见图 6-4）。

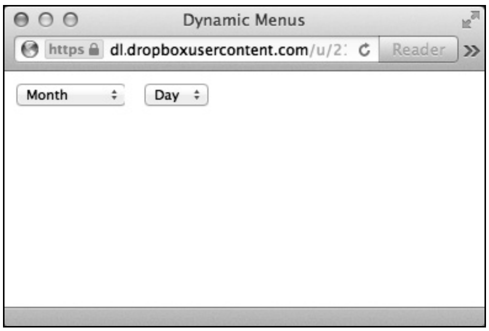


图 6-3　当用户在 Month 菜单中作出选择时，会自动地填充 Day 菜单的内容



图 6-4　选择一个月份的结果：Day 菜单中显示这个月正确的天数

脚本 6-4　弹出菜单的 HTML 列出了月份，但是没有列出天数

```
<!DOCTYPE html>
<html>
<head>
     <title>Dynamic Menus</title>
     <script src="script02.js"></script>
</head>
<body>
<form action="#">
     <select id="months">
        <option value="">Month</option>
```

```
        <option value="0">January</option>
        <option value="1">February</option>
        <option value="2">March</option>
        <option value="3">April</option>
        <option value="4">May</option>
        <option value="5">June</option>
        <option value="6">July</option>
        <option value="7">August</option>
        <option value="8">September</option>
        <option value="9">October</option>
        <option value="10">November</option>
        <option value="11">December</option>
     </select>
      
     <select id="days">
        <option>Day</option>
     </select>
</form>
</body>
</html>
```


脚本 6-5 在弹出菜单中选择一个值，然后就可以创建第二个弹出菜单的内容

```
window.onload = initForm;

function initForm() {
     document.getElementById("months").selectedIndex = 0;
     document.getElementById("months").onchange = populateDays;
}

function populateDays() {
     var monthDays = new Array(31,28,31,30,31,30,31,31,30,31,30,31);
     var monthStr = this.options[this.selectedIndex].value;

     if (monthStr != "") {
        var theMonth = parseInt(monthStr);

        document.getElementById("days").options.length = 0;
        for (var i=0; i<monthDays[theMonth];i++) {
           document.getElementById("days").options[i] = new Option(i+1);
        }
     }
}
```

⇛ 动态地改变菜单

1. `var monthDays = new Array(31,28,31,30,31,30,31,31,30,31,30,31);`

这个新数组包含对应于 12 个月的 12 个数字，即每个月的正确天数。这个数组存储在变量 monthDays 中。

2. `var monthStr = this.options[this.selectedIndex].value;`

我们使用 this（用户在第一个菜单中选择的月份）从菜单中获得值，并且将它存储在 monthStr 中。

3.
```
if (monthStr != "") {
   var theMonth = parseInt(monthStr);
```

如果 monthStr 的值是""，那么用户就是在菜单中选择了单词 Month，而不是月份名。这些代码的作用是检查 monthStr 的值是否不是""，如果这个条件为真，那么用 parseInt 方法将 monthStr 转换为数字，并且将变量 theMonth 设置为这个结果。

4. `document.getElementById("days").options.length = 0;`

```
for(var i=0; i<monthDays[theMonth]; i++) {
  document.getElementById("days").options[i] = new Option(i+1);
```

在改变 Day 菜单时，首先将它的选项长度设置为零，这会消除以前操作的影响。这个循环简单地遍历所选月份中的每一天，为每一天在菜单中添加一个新选项。传递给 Option 的是 `i+1`，这使它显示 1～31，而不是 0～30。

✔提示

- ❑ `monthDays` 数组包含每个月的天数，如果不是闰年的话，这种方式就没问题。要想让脚本能够处理闰年，就需要改变 `monthDays` 中二月的值。

6.3　建立必须填写的字段

在填写表单时，可能希望指定用户必须填写某些字段，然后才能提交表单。可以使用 JavaScript 检查某些或所有字段是否已经填写了。在这个示例中，我们使用 HTML、CSS 和 JavaScript（分别是脚本 6-6、脚本 6-7 和脚本 6-8）通过红色的边框和黄色的内部颜色突出显示未填写的字段。检查在用户单击表单的 Submit 按钮时进行。

脚本 6-6　密码检查示例的 HTML

```
<!DOCTYPE html>
<html>
<head>
     <title>Password Check</title>
     <link rel="stylesheet" href="script03.css">
     <script src="script03.js"></script>
</head>
<body>
<form action="#">
     <p><label for="userName">Your name:<input type="text" size="30" id="userName" class="reqd"></label> </p>
     <p><label for="passwd1">Choose a password: <input type="password" id="passwd1" class="reqd"></label></p>
     <p><label for="passwd2">Verify password:<input type="password" id="passwd2" class="reqd passwd1">
     →</label></p>
     <p><input type="submit" value="Submit"> <input type="reset"></p>
</form>
</body>
</html>
```

脚本 6-7　这段 CSS 为无效的表单元素设置样式

```
body {
     color: #000;
     background-color: #FFF;
}

input.invalid {
     background-color: #FF9;
     border: 2px red inset;
}

label.invalid {
```

```
        color: #F00;
        font-weight: bold;
}
```

脚本 6-8 这个脚本是本章中其余所有示例的基础，可以使用这个框架添加额外的有效性检查

```
window.onload = function() {
     document.forms[0].onsubmit = validForm;
}

function validForm() {
     var allGood = true;
     var allTags = document.forms[0].getElementsByTagName("*");

     for (var i=0; i<allTags.length; i++) {
        if (!validTag(allTags[i])) {
           allGood = false;
        }
     }
     return allGood;

     function validTag(thisTag) {
        var outClass = "";
        var allClasses = thisTag.className.split(" ");

        for (var j=0; j<allClasses.length;j++) {
           outClass += validBasedOnClass(allClasses[j]) + " ";
        }

        thisTag.className = outClass;

        if (outClass.indexOf("invalid") > -1) {
           thisTag.focus();
           if (thisTag.nodeName == "INPUT") {
              thisTag.select();
           }
           return false;
        }
        return true;

        function validBasedOnClass(thisClass) {
           var classBack = "";

           switch(thisClass) {
              case "":
              case "invalid":
                 break;
              case "reqd":
                 if (allGood && thisTag.value == "") {
                    classBack = "invalid ";
                 }
                 classBack += thisClass;
                 break;
              default:
                 classBack += thisClass;
           }
           return classBack;
        }
     }
}
```

6

这种检查方法的基本原理如下：HTML 中的 class 属性表示我们希望 JavaScript 执行哪种检查。如果检查失败，就在 class 属性的列表中添加 invalid。这样做会出现两种现象：(1) 表单提交失败；(2) 脚本 6-7 中的 CSS 改变这个字段在页面上的外观（见图 6-5）。



图 6-5　通过突出显示背景色让用户知道某个字段有问题，从而确保用户正确输入密码

⇒ 建立必须填写的字段

1. window.onload = function() {

```
document.forms[0].onsubmit = validForm;
```

当页面首次加载时，匿名函数会查找页面中的第一个表单。对于这个表单，它为表单的 onsubmit 添加一个事件处理程序：调用 validForm。当 onsubmit 处理程序返回 false 值时，表单就不会被传递回服务器。只有在返回 true 值时，服务器才会收到表单（因此，运行动作属性中存储的服务器端脚本）。

2. var allTags = document.forms[0]getElementsByTagName("*");

document.getElementsByTagName("*")是非常有用的——星号让 JavaScript 返回一个包含页面上所有标签的数组。如果能使用这个函数获取特定元素内的所有标签就更方便了，此处就是这种情况。这一行代码查找页面第一个表单内部的所有标签。然后，就可以遍历 allTags 数组来寻找感兴趣的标签。

3. for (var i=0; i<allTags.length;i++) {

```
if (!validTag(allTags[i])) {
  allGood = false;
```

这个循环在 allTags 中进行搜索，而 if 条件语句调用 validTag()函数，这会检查每个标签，了解是否有什么东西阻止表单提交这个页面。传递给 validTag()函数的是 allTags[i]，也就是当前正在处理的对象。如果任何标签导致 validTag()返回 false，就将 allGood 设置为 false。但是，即使已经出现了 false，我们仍然继续循环遍历所有标签。

4. return allGood;

现在返回 allGood，以此表明是否应该继续提交表单。

5. function validTag(thisTag) {

创建 validTag()函数，并且让它接收 thisTag 值。

6. var allClasses = thisTag.className.split(" ");

对于每个标签，我们希望检查每个 class 属性（请记住，class 可以设置为多个属性）。创建

allClasses 数组并且用 thisTag.className.split(" ")设置它。这个函数会按照传递给它的字符串，将字符串分割为数组。在这里，分割符字符串是一个空格，所以它会将字符串"this that and the other"转换为一个包含 5 个元素的数组：this、that、and、the 和 other。

我们希望检查每个 class 属性，因为 class 表示我们希望对每个表单字段进行什么处理。在这个示例中，我们关心的一个值是 reqd，这表示必须填写这个字段。如果任何表单字段具有 class 值 reqd，那么它必须包含某些值。

7.
```
for (var j=0; j<allClasses.length;j++) {
    outClass += validBasedOnClass(allClasses[j]) + " ";
```

这个循环使用 j 作为循环变量，因为外层的循环已经使用了 i。对于 allClasses 中的每个 class 属性循环一次。

对于每个类，执行 outClass += validBasedOnClass(allClasses[j]) + " ";。这会调用 validBasedOnClass()函数（后面会解释这个函数），并且传递当前查看的类。这个函数返回某些东西，这些东西加上一个空格之后追加到 outClass 变量中。

8.
```
thisTag.className = outClass;
```

在结束 allClasses 循环时，我们获得 outClass 的内容并且将它放进 thisTag.className，这会覆盖这个表单字段当前的 class 属性。这是因为在进行处理期间 class 可以改变，稍后就会看到这种情况。

9.
```
if (outClass.indexOf("invalid") > -1) {
```

在新的 class 属性中可能返回的值之一是单词 invalid，所以要检查它。如果在新的类中找到了 invalid，就说明有问题，因此执行对应的操作。

10.
```
thisTag.focus();
```

如果这个表单字段可以获得焦点（还记得吗，我们在第 5 章中讨论过焦点），那么希望将焦点放进这个字段，这一行就完成这个任务。这是让用户知道哪个字段有问题的一种方法。

11.
```
if (thisTag.nodeName == "INPUT") {
    thisTag.select();
```

这些代码的基本意思是，“当前查看的这个标签是<input>标签吗？如果是，就选择它的值，让用户能够更轻松地修改它。”

12.
```
return false;
```

我们仍然在“返回无效元素”块中，所以将 false 返回到进行调用的地方。

13.
```
return true;
```

如果所有一切都运行良好并且有效，就返回 true。

14.
```
function validBasedOnClass(thisClass) {
    var classBack = "";
```

创建新的 validBasedOnClass()函数，并且让它接收 thisClass 值。接下来创建 classBack 变量，并且将它初始化为空。这个变量将包含要返回的类，也就是我们希望发送回的值。

15.
```
switch(thisClass) {
```

switch 语句检查（在 thisClass 中）传递给它的一个 class 属性，并根据这个属性执行对应的操作。

```
16. case "":
    case "invalid":
      break;
```

如果 thisClass 是空的或是 invalid，那么就跳出条件语句，否则继续。

```
17. case "reqd":
      if (allGood && thisTag.value == ""){
         classBack = "invalid ";
      }
      classBack += thisClass;
      break;
```

如果正在处理的属性是 reqd，allGood 是 true，而且当前标签的当前值是""（即空字符串），那么将 classBack 设置为 invalid，因为这说明有问题，我们希望通知用户。在此之后，无论是否有问题，我们都将当前的类追加到 classBack 中，使它不会丢失。

```
18. default:
      classBack += thisClass;
```

如果上面的分支与发生的情况都不匹配，就会执行 default 块。当发生这种情况时，就说明当前处理的类是我们不关心的，所以只需要将它追加到 classBack 中，而不必担心其他事情。

```
19. return classBack;
```

最后，返回 classBack。

✔提示

- ❑ 好的 UI 设计只会在一个页面上设置一个表单，因此我们这里的脚本是按照只需要查看第一个也是唯一一个表单来的。如果你想为页面添加多个网页验证表单，那么别忘记修改脚本。

6.4　根据其他字段对字段进行检查

经常需要根据另一个字段对一个字段进行检查，尤其是在要求用户设置密码时。为了确保密码正确，希望用户输入密码两次，并确保两次的输入完全相同。

这个示例重用了脚本 6-6（HTML）和脚本 6-7（CSS），而且只需在脚本 6-8 中添加几行 JavaScript（见脚本 6-9），使这个脚本具有额外的核对功能。结果见图 6-6。同样，当检查失败时，无效的字段会显示红色的边框。



图 6-6　对两个密码字段进行核对，确保它们的内容是相同的。如果不匹配，就显示图中的效果

脚本 6-9 使用这个脚本比较两个字段的值，检查它们是否匹配

```
window.onload = function() {
     document.forms[0].onsubmit = validForm;
}

function validForm() {
     var allGood = true;
     var allTags = document.forms[0].getElementsByTagName("*");

     for (var i=0; i<allTags.length; i++) {
        if (!validTag(allTags[i])) {
           allGood = false;
        }
     }
     return allGood;

     function validTag(thisTag) {
        var outClass = "";
        var allClasses = thisTag.className.split(" ");

        for (var j=0; j<allClasses.length; j++) {
           outClass += validBasedOnClass(allClasses[j]) + " ";
        }

        thisTag.className = outClass;

        if (outClass.indexOf("invalid") > -1) {
           thisTag.focus();
           if (thisTag.nodeName == "INPUT") {
              thisTag.select();
           }
           return false;
        }
        return true;

        function validBasedOnClass(thisClass) {
           var classBack = "";

           switch(thisClass) {
              case "":
              case "invalid":
                 break;
              case "reqd":
                 if (allGood && thisTag.value == "") {
                    classBack = "invalid ";
                 }
                 classBack += thisClass;
                 break;
              default:
                 if (allGood && !crossCheck(thisTag,thisClass)) {
                    classBack = "invalid ";
                 }
                 classBack += thisClass;
```

```
            }
            return classBack;

            function crossCheck(inTag,otherFieldID) {
               if (!document.getElementById(otherFieldID)) {
                  return false;
               }
               return (inTag.value == document.getElementById(otherFieldID).value);
            }
         }
      }
   }
```

⇛ 根据另一个字段对一个字段进行检查

```
1. if (allGood && !crossCheck(thisTag,thisClass)) {
      classBack = "invalid ";
```

现在，我们要检查两个密码字段是否相同。因为第二个密码字段的类包含 passwd1（见脚本 6-6），所以这个 JavaScript 脚本知道它必须根据第一个密码字段核对第二个密码字段。这在条件语句的 default 块中进行处理。如果 allGood 是 true 且 crossCheck() 函数（见下面）发现了问题（并返回 false），那么就将 classBack 设置为 invalid。

```
2. function
   crosscheck (inTag,otherFieldID) {
     if (!document.getElementById(otherFieldID)) {
        return false;
     }
     return (inTag.value == document.getElementById(otherFieldID).value);
```

这是 crossCheck() 函数。它接收当前标签和检查所针对的另一个字段的 id。在这个示例中，当前标签是 passwd2 <input>，另一个字段的 id 是 passwd1。如果另一个字段不存在，就无法进行检查，这就说明有问题，所以函数返回 false。否则，这两个字段都存在，所以可以比较它们的值：如果相同，就返回 true；如果不同，就返回 false。

✔提示

- ❑ 这个脚本并没有根据主密码数据库检查用户输入的密码是否有效，这种任务需要服务器端脚本来实现，它只是确保用户两次输入的密码是相同的。

6.5 标识有问题的字段

将输入字段的边框改为红色是很不错的，但是如果能够让有问题的字段再醒目一点儿就更好了。在这个示例中，你将学习如何将字段旁边的标签设置为红色的粗体字，从而使出问题的字段更加明显（见图 6-7）。同样，HTML 和 CSS 文件没有改动（仍然是脚本 6-6 和脚本 6-7）。在脚本 6-10 中，我们在脚本 6-9 中添加了几行 JavaScript 代码以进一步突出显示输入错误。

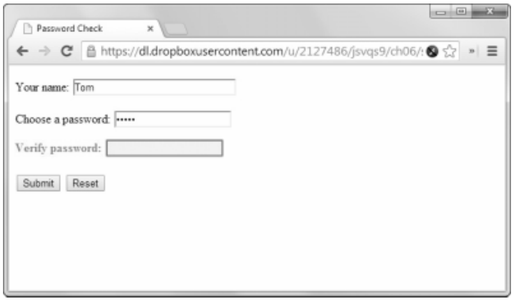


图 6-7　当出现问题时，可以让字段的标签成为红色的粗体字，字段本身也改变样式

脚本 6-10　当发现错误时，这个脚本会突出显示错误的字段的标签


```
window.onload = function() {
     document.forms[0].onsubmit = validForm;
}

function validForm() {
     var allGood = true;
     var allTags = document.forms[0].getElementsByTagName("*");

     for (var i=0; i<allTags.length; i++) {
        if (!validTag(allTags[i])) {
           allGood = false;
        }
     }
     return allGood;

     function validTag(thisTag) {
        var outClass = "";
        var allClasses = thisTag.className.split(" ");

        for (var j=0; j<allClasses.length; j++) {
           outClass += validBasedOnClass(allClasses[j]) + " ";
        }

        thisTag.className = outClass;

        if (outClass.indexOf("invalid") > -1) {
           invalidLabel(thisTag.parentNode);
           thisTag.focus();
           if (thisTag.nodeName == "INPUT") {
              thisTag.select();
           }
           return false;
        }
        return true;

        function validBasedOnClass(thisClass) {
           var classBack = "";

           switch(thisClass) {
              case "":
              case "invalid":
                 break;
```

```
                case "reqd":
                    if (allGood && thisTag.alue == "") {
                        classBack = "invalid ";
                    }
                    classBack += thisClass;
                    break;
                default:
                    if (allGood && !crossCheck(thisTag,thisClass)) {
                        classBack = "invalid ";
                    }
                    classBack += thisClass;
            }
            return classBack;

            function crossCheck(inTag,otherFieldID) {
                if (!document.getElementById(otherFieldID)) {
                    return false;
                }
                return (inTag.value == document.getElementById(otherFieldID).value);
            }
        }
    }

    function invalidLabel(parentTag) {
        if (parentTag.nodeName == "LABEL") {
            parentTag.className += " invalid";
        }
    }
}
```

⇛ 标识有问题的表单字段

1. `invalidLabel(thisTag.parentNode);`

这行代码被添加到 validTag()内的无效检查中。如果当前字段没有通过有效性检查，那么我们希望检查是否能够使包围这个元素的标签也成为无效的。为此，要调用新的 invalidLabel()函数（下面有解释）并且将当前标签的父标签传递给它。也就是说，如果 passwd1 输入字段出了问题，那么我们希望给这个标签和包围它的 label 标签都分配 class 属性 invalid。所以，一旦发现 passwd1 输入字段出了问题，就将它的父标签（label 标签）传递给 invalidLabel()，从而检查是否能够将它标记为无效的。

2.
```
function invalidLabel(parentTag) {
    if (parentTag.nodeName == "LABEL") {
      parentTag.className += " invalid";
```

这个函数接受一个标签并且检查这个标签（tag）是否是一个标签（label）。如果是，就在它的类中添加 invalid 属性。

现在，如果试图提交表单，而表单上有错误，就会看到有问题的字段的标签变成了红色的粗体字。纠正错误，再次提交表单，标签就会恢复为黑色。

6.6 准备进行表单验证

关于前几个示例中构建的脚本，有意思的一点是：脚本在很大程度上独立于使用它的 HTML 页面。

换句话说，可以改用一个完全不同的页面，其中包含完全不同的表单，而且只需对脚本略作修改，就可以执行你需要的所有验证任务。

例如，看一下图 6-8，这是一个表单的简化版本，用户可以使用它定制要购买的汽车。这个表单包含各种选项和界面元素，包括单选按钮、菜单、复选框和文本字段，它们都需要检查数据输入是否正确。这个表单的 HTML 见脚本 6-11，CSS 见脚本 6-12。在本章的其余示例中，我们将一直使用这个 HTML。



图 6-8 Car Picker 表单使用了文本字段、弹出菜单、复选框和单选按钮，这些都是常用的表单元素

脚本 6-11 这是 Car Picker 示例的完整 HTML 代码页面

```
<!DOCTYPE html>
<html>
<head>
     <title>Car Picker</title>
     <link rel="stylesheet" href="script06.css">
     <script src="script06.js"></script>
</head>
<body>
<h2 class="centered">Car Picker</h2>
<form action="someAction.cgi">
     <p><label for="emailAddr">Email Address: <input id="emailAddr" type="text" size="30" class="reqd email">
     →</label></p>
     <p>Colors:
        <select id="color" class="reqd">
           <option value="" selected>Choose a color</option>
           <option value="Red">Red</option>
           <option value="Green">Green</option>
           <option value="Blue">Blue</option>
        </select>
     </p>
     <p>Options:
        <label for="sunroof"><input type="checkbox" id="sunroof" value="Yes">Sunroof (Two door only)</label>
        <label for="pWindows"><input type="checkbox" id="pWindows" value="Yes">Power Windows</label>
     </p>
```

```
        <p>Doors:  
           <label for="twoDoor"><input type="radio" id="twoDoor" name="DoorCt" value="twoDoor" class="radio">
           ➝Two</label>
           <label for="fourDoor"><input type="radio" id="fourDoor" name="DoorCt" value="fourDoor" class="radio">
           ➝Four</label>
        </p>
        <p>
           <label for="zip">Enter your Zip code or pick the dealer nearest you:<br>
              Zip: <input id="zip" type="text" size="5" maxlength="5" class="isZip dealerList">
           </label>
           <select id="dealerList" size="4" class="zip">
              <option value="California--Lemon Grove">California--Lemon Grove</option>
              <option value="California--Lomita">California--Lomita</option>
              <option value="California--Long Beach">California--Long Beach</option>
              <option value="California--Los Alamitos">California--Los Alamitos</option>
              <option value="California--Los Angeles">California--Los Angeles</option>
           </select>
        </p>
        <p><input type="submit" value="Submit"> <input type="reset"></p>
   </form>
   </body>
   </html>
```

脚本 6-12 这里显示的某些样式只能以后使用

```
body {
     color: #000;
     background-color: #FFF;
}

input.invalid {
     background-color: #FF9;
     border: 2px red inset;
}

label.invalid {
     color: #F00;
     font-weight: bold;
}

select {
     margin-left: 80px;
}

input {
     margin-left: 30px;
}

input+select, input+input {
     margin-left: 20px;
}

.centered {
     text-align: center;
}
```

JavaScript 文件（见脚本 6-13）是在本章前面使用的脚本基础上构建的。我们在脚本中添加了几行代码来处理新的界面元素，脚本的其他部分是相同的。在这个示例中，你将看到为了进行更多验证，需要在脚本中添加什么代码，而后续的示例将更深入地研究特定类型的表单元素。

脚本 6-13 这个脚本在 switch/case 条件结构中添加了几个代码块，后面的示例将利用这些结构

```
window.onload = function() {
     document.forms[0].onsubmit = validForm;
}

function validForm() {
     var allGood = true;
     var allTags = document.forms[0].getElementsByTagName("*");

     for (var i=0; i<allTags.length; i++) {
        if (!validTag(allTags[i])) {
           allGood = false;
        }
     }
     return allGood;

     function validTag(thisTag) {
        var outClass = "";
        var allClasses = thisTag.className.split(" ");

        for (var j=0; j<allClasses.length; j++) {
           outClass += validBasedOnClass(allClasses[j]) + " ";
        }

        thisTag.className = outClass;

        if (outClass.indexOf("invalid") > -1) {
           invalidLabel(thisTag.parentNode);
           thisTag.focus();
           if (thisTag.nodeName == "INPUT") {
              thisTag.select();
           }
           return false;
        }
        return true;

        function validBasedOnClass(thisClass) {
           var classBack = "";

           switch(thisClass) {
              case "":
              case "invalid":
                 break;
              case "reqd":
                 if (allGood && thisTag.value == "") {
                    classBack = "invalid ";
                 }
                 classBack += thisClass;
                 break;
              case "radio":
              case "isNum":
              case "isZip":
              case "email":
                 classBack += thisClass;
                 break;
              default:
                 if (allGood && !crossCheck(thisTag,thisClass)) {
                    classBack = "invalid ";
                 }
```

6

```
                classBack += thisClass;
            }
            return classBack;

            function crossCheck(inTag,otherFieldID) {
               if (!document.getElementById(otherFieldID)) {
                  return false;
               }
               return (inTag.value != "" || document.getElementById(otherFieldID).value != "");
            }
         }
      }

      function invalidLabel(parentTag) {
         if (parentTag.nodeName == "LABEL") {
            parentTag.className += " invalid";
         }
      }
   }
```

⇒ 对具有许多元素的表单进行验证

```
1. case "radio":
   case "isNum":
   case "isZip":
   case "email":
      classBack += thisClass;
      break;
```

通过在 validBasedOnClass()函数的 switch/case 条件结构中添加额外的代码块，我们使这个脚本能够检查更多的字段和更多的情况。我们在分支列表中添加了 radio、isNum、isZip 和 email。尽管在这个示例中不对它们进行验证，但是在以后进行验证时，我们不希望这些结构造成问题，所以对于每个还未处理的属性，要在 switch/case 处理的属性列表中保留它们。因为前 3 个代码块中没有指令，所以它们都会执行 email 分支中的代码，这里的代码仅仅将当前检查的属性添加到 classBack 中。

```
2. return (inTag.value != "" || document.getElementById(otherFieldID).value != "");
```

crossCheck()中的这一行做了一点儿改动。这里不再比较两个字段是否相等，而是检查是否至少已经设置了两个字段之一（这是为表单末尾的 ZIP 编码字段和列表元素准备的）。如果其中至少一个字段包含值，那么返回 true；如果它们都不包含值，那么返回 false。

6.7　处理单选按钮

单选按钮是一种界面元素，它告诉用户在一组选项中选择一个选项（而且只能选择一个）。如果需要从多个选项中选择一个，就应该使用单选按钮。如图 6-9 所示，这个表单让假定购买汽车的顾客用单选按钮选择两门汽车或四门汽车。在这种情况下，只能选择这些选项之一，而且必须作出选择。



图 6-9　单选按钮是让用户在一组选项中只选择一个选项的最佳方法

如脚本 6-14 所示，检查是否选择了一个按钮并不需要很多脚本代码。我们采用的方法是，遍历每个按钮并且检查它的状态，如果没有选择按钮，就将单选按钮的标签和按钮本身转换为红色的粗体。

脚本 6-14　只能选择一个单选按钮，而且这段 JavaScript 会执行界面规则

```
window.onload = function() {
     document.forms[0].onsubmit = validForm;
}

function validForm() {
     var allGood = true;
     var allTags = document.forms[0].getElementsByTagName("*");

     for (var i=0; i<allTags.length; i++) {
        if (!validTag(allTags[i])) {
           allGood = false;
        }
     }
     return allGood;

     function validTag(thisTag) {
        var outClass = "";
        var allClasses = thisTag.className.split(" ");

        for (var j=0; j<allClasses.length; j++) {
           outClass += validBasedOnClass(allClasses[j]) + " ";
        }

        thisTag.className = outClass;

        if (outClass.indexOf("invalid") > -1) {
           invalidLabel(thisTag.parentNode);
           thisTag.focus();
           if (thisTag.nodeName == "INPUT") {
              thisTag.select();
           }
           return false;
        }
        return true;

        function validBasedOnClass(thisClass) {
           var classBack = "";

           switch(thisClass) {
              case "":
              case "invalid":
                 break;
              case "reqd":
                 if (allGood && thisTag.value == "") {
                    classBack = "invalid ";
                 }
                 classBack += thisClass;
                 break;
              case "radio":
                 if (allGood && !radioPicked(thisTag.name)) {
                    classBack = "invalid ";
                 }
                 classBack += thisClass;
                 break;
```

```
            case "isNum":
            case "isZip":
            case "email":
               classBack += thisClass;
               break;
            default:
               if (allGood && !crossCheck(thisTag,thisClass)) {
                  classBack = "invalid ";
               }
               classBack += thisClass;
         }
         return classBack;

         function crossCheck(inTag,otherFieldID) {
            if (!document.getElementById(otherFieldID)) {
               return false;
            }
            return (inTag.value != "" || document.getElementById(otherFieldID).value != "");
         }

         function radioPicked(radioName) {
            var radioSet = document.forms[0][radioName];

            if (radioSet) {
               for (k=0; k<radioSet.length; k++) {
                  if (radioSet[k].checked) {
                     return true;
                  }
               }
            }
            return false;
         }
      }
   }

   function invalidLabel(parentTag) {
      if (parentTag.nodeName == "LABEL") {
         parentTag.className += " invalid";
      }
   }
}
```

⇛ 确保用户选择一个单选按钮

1. `if (allGood && !radioPicked(thisTag.name)){`
   `classBack = "invalid ";`

这行代码在 switch/case 条件结构的 radio 块中。我们希望确保至少选择一个单选按钮，新的 radioPicked()函数处理这个问题。如果它返回 false，那么就将 classBack 设置为 invalid。

2. `function radioPicked(radioName) {`
   `var radioSet = document.forms[0][radioName]`

开始新的 radioPicked()函数，并且对 radioSet 变量进行初始化。

这个函数接受单选按钮组的名称，这个示例中就是 DoorCt，见脚本 6-11。注意，这不是当前标签的 id、class 或我们常常看到的任何东西，而是它的 name。<input>标签的 name 属性使 HTML 知道哪些单选按钮分组在一起。也就是说，所有具有相同 name 属性的<input>标签都属于同一个单选按钮组。

然后，尝试将 radioSet 设置为正在查看的表单中单选按钮组的名称。如果找到这个单选按钮组，radioSet 就会包含一个值。

```
3. if (radioSet) {
     for (k=0; k<radioSet.length; k++) {
       if (radioSet[k].checked) {
         return true;
```

如果 radioSet 包含一个值，那我们就知道了想要检查的单选按钮组。接下来启用另一个循环检查每个单选按钮。当发现一个单选按钮被选中时，就返回 true，因为这就是我们希望保证的。

```
4. return false;
```

如果到达了循环末尾的这个语句，那么已经检查了整个单选按钮组而且没有单选按钮被选中。在这种情况下，返回 false，并且改变单选按钮的标签并将单选按钮的文本改为红色的粗体。

6.8　用一个字段设置另一个字段

在表单上常见的一种情况是，如果用户作出选择，那么这个选择会影响表单上其他字段的值。例如，假设只有在两门汽车上才能选择遮阳篷（sunroof）选项。处理这个问题有两种方法。第一种方法是检查输入，如果用户作出了错误的选择，就弹出警告对话框。但是，更好的方法是替用户输入。所以，如果用户选择了遮阳篷，脚本就会自动选中两门选项，如图 6-10 所示。脚本 6-15 演示了具体做法。

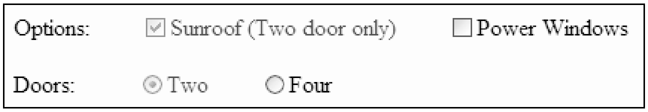


图 6-10　当用户选择了遮阳篷选项时，脚本会自动设置两门单选按钮

脚本 6-15　处理用户选择的一种好方法是，根据用户作出的其他选择自动设置字段输入

```
window.onload = initForm;

function initForm() {
     document.forms[0].onsubmit = validForm;
     document.getElementById("sunroof").onclick = doorSet;
}

function validForm() {
     var allGood = true;
     var allTags = document.forms[0].getElementsByTagName("*");

     for (var i=0; i<allTags.length; i++) {
        if (!validTag(allTags[i])) {
           allGood = false;
        }
     }
     return allGood;

     function validTag(thisTag) {
        var outClass = "";
```

```
var allClasses = thisTag.className.split(" ");

for (var j=0; j<allClasses.length; j++) {
   outClass += validBasedOnClass(allClasses[j]) + " ";
}

thisTag.className = outClass;

if (outClass.indexOf("invalid") > -1) {
   invalidLabel(thisTag.parentNode);
   thisTag.focus();
   if (thisTag.nodeName == "INPUT") {
      thisTag.select();
   }
   return false;
}
return true;

function validBasedOnClass(thisClass) {
   var classBack = "";

   switch(thisClass) {
      case "":
      case "invalid":
         break;
      case "reqd":
         if (allGood && thisTag.value == "") {
            classBack = "invalid ";
         }
         classBack += thisClass;
         break;
      case "radio":
         if (allGood && !radioPicked(thisTag.name)) {
            classBack = "invalid ";
         }
         classBack += thisClass;
         break;
      case "isNum":
      case "isZip":
      case "email":
         classBack += thisClass;
         break;
      default:
         if (allGood && !crossCheck(thisTag,thisClass)) {
            classBack = "invalid ";
         }
         classBack += thisClass;
   }
   return classBack;

   function crossCheck(inTag,otherFieldID) {
      if (!document.getElementById(otherFieldID)) {
         return false;
      }
      return (inTag.value != "" || document.getElementById(otherFieldID).value != "");
   }

   function radioPicked(radioName) {
      var radioSet = document.forms[0][radioName];

      if (radioSet) {
```

```
                    for (k=0; k<radioSet.length; k++) {
                       if (radioSet[k].checked) {
                          return true;
                       }
                    }
                 }
                 return false;
              }
           }
        }

        function invalidLabel(parentTag) {
           if (parentTag.nodeName == "LABEL") {
              parentTag.className += " invalid";
           }
        }
    }

    function doorSet() {
        if (this.checked) {
           document.getElementById("twoDoor").checked = true;
        }
    }
```


⇛ 自动设置字段值

1. `document.getElementById("sunroof").onclick = doorSet;`

这行代码添加在 `initForm()`中。当用户单击遮阳篷复选框时，将调用 `doorSet()`函数。

2.
```
function doorSet() {
    if (this.checked) {
      document.getElementById("twoDoor").checked = true;
```

这个新函数检查是否选中了遮阳篷选项。如果是，就将 `twoDoor` 单选按钮设置为 `true`。如果是取消选中遮阳篷复选框，那么不做任何处理。

✔提示

- ❑ 你可能会注意到，这里没有检查用户是否取消选中遮阳篷选项，然后重新设置 fourDoor 单选按钮。这个功能留给读者作为练习。

6.9 检验 Zip 编码

那些古怪的用户可能在表单中输入几乎任何东西，所以需要确保用户在 Zip 编码字段中输入的内容只包含数字（见图 6-11）。脚本 6-16 演示具体做法。

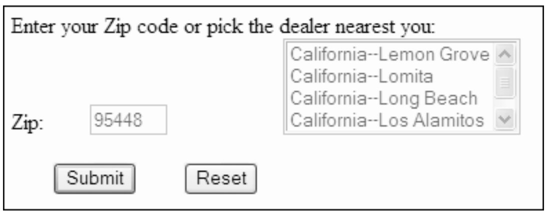


图 6-11　可以确保用户输入了 Zip 编码或者从滚动列表中选择

脚本 6-16　只需几行 JavaScript 就可以从 Zip 编码中消除错误的字符

```
window.onload = initForm;

function initForm() {
     document.forms[0].onsubmit = validForm;
     document.getElementById("sunroof").onclick = doorSet;
}

function validForm() {
     var allGood = true;
     var allTags = document.forms[0].getElementsByTagName("*");

     for (var i=0; i<allTags.length; i++) {
        if (!validTag(allTags[i])) {
           allGood = false;
        }
     }
     return allGood;

     function validTag(thisTag) {
        var outClass = "";
        var allClasses = thisTag.className.split(" ");

        for (var j=0; j<allClasses.length; j++) {
           outClass += validBasedOnClass(allClasses[j]) + " ";
        }

        thisTag.className = outClass;

        if (outClass.indexOf("invalid") > -1) {
           invalidLabel(thisTag.parentNode);
           thisTag.focus();
           if (thisTag.nodeName == "INPUT") {
              thisTag.select();
           }
           return false;
        }
        return true;

        function validBasedOnClass(thisClass) {
           var classBack = "";

           switch(thisClass) {
              case "":
              case "invalid":
                 break;
              case "reqd":
                 if (allGood && thisTag.value == "") {
                    classBack = "invalid ";
                 }
                 classBack += thisClass;
                 break;
              case "radio":
                 if (allGood && !radioPicked(thisTag.name)) {
                    classBack = "invalid ";
                 }
                 classBack += thisClass;
                 break;
              case "isNum":
```

```
            if (allGood && !isNum(thisTag.value)) {
               classBack = "invalid ";
            }
            classBack += thisClass;
            break;
         case "isZip":
            if (allGood && !isZip(thisTag.value)) {
               classBack = "invalid ";
            }
            classBack += thisClass;
            break;
         case "email":
            classBack += thisClass;
            break;
         default:
            if (allGood && !crossCheck(thisTag,thisClass)) {
               classBack = "invalid ";
            }
            classBack += thisClass;
      }
      return classBack;

      function crossCheck(inTag,otherFieldID) {
         if (!document.getElementById(otherFieldID)) {
            return false;
         }
         return (inTag.value != "" || document.getElementById(otherFieldID).value != "");
      }

      function radioPicked(radioName) {
         var radioSet = document.forms[0][radioName];

         if (radioSet) {
            for (k=0; k<radioSet.length;k++) {
               if (radioSet[k].checked) {
                  return true;
               }
            }
         }
         return false;
      }

      function isNum(passedVal) {
         if (passedVal == "") {
            return false;
         }
         for (var k=0;  k<passedVal.length; k++) {
            if (passedVal.charAt(k) < "0") {
               return false;
            }
            if (passedVal.charAt(k) > "9") {
               return false;
            }
         }
         return true;
      }
      function isZip(inZip) {
         if (inZip == "") {
            return true;
         }
```

```
                return (isNum(inZip));
            }
        }
    }

    function invalidLabel(parentTag) {
        if (parentTag.nodeName == "LABEL") {
            parentTag.className += " invalid";
        }
    }
}

function doorSet() {
    if (this.checked) {
        document.getElementById("twoDoor").checked = true;
    }
}
```

⇒ 确保 Zip 编码是正确的

1. `if (allGood && !isNum(thisTag.value)) {`
```
    classBack = "invalid ";
}
classBack += thisClass;
```
这些代码放在 switch/case 条件结构的 isNum 块中。如果输入的是非数字,那么 isNum()返回 false。

2. `if (allGood && !isZip(thisTag.value)) {`
```
    classBack = "invalid ";
```
这行代码添加在 switch/case 条件结构的 isZip 块中。如果这个字段非空而且它不是 Zip 编码，那么 isZip()返回 false。

3. `if (passedVal == "") {`
```
    return false;
```
在 isNum()函数中，如果 passedVal 是空的，那么我们正在检查的字段就不是数字。当发生这种情况时，返回 false，表示出现了错误。

4. `for (var k=0; k<passedVal.length; k++) {`

现在，建立一个循环，每次迭代将 k 计数器递增，直到 passedVal 的长度。使用 k 是因为目前我们已经处于另两个循环（i 和 j）之中。

5. `if (passedVal.charAt(k) < "0") {`
```
    return false;
}
if (passedVal.charAt(k) > "9") {
    return false;
```
charAt()函数检查位置 k 上的字符。如果这个字符小于“0”或者大于“9”，那么它就不是数字，所以要返回 false，表示输入是非数字。

6. `return true;`

如果到达这行代码，就说明输入是数字，所以返回 true。

7.
```
function isZip(inZip) {
    if (inZip == "") {
      return true;
    }
    return (isNum(inZip));
```

在这个表单中，Zip 编码字段为空是有效的。因此我们先检查用户是否在这个字段中输入了任何内容，如果没有，就返回 true——这是有效的输入。但是，如果输入了任何内容，它就必须是数字，所以用 isNum()进行检查。

✔提示

- ❑ 如果以后希望在 HTML 表单中添加另一个必须是数字的新字段，那么不需要编写新的 JavaScript 代码，只需使用现有的 isNum 检查。
- ❑ 请记住，网络是全球性的，用户并非都是美国人。如果你的站点希望吸引美国之外的用户，那么不要要求用户输入 Zip 编码。美国之外的地址可能有邮政编码（postal code），也可能没有，而且这些邮政编码可能不是数字。

6.10　验证电子邮件地址

用户在输入因特网地址方面可能会遇到困难，尤其是新用户。你可以扫描他们输入的电子邮件地址并检查其形式是否正确，从而帮助用户输入正确的地址。例如，可以检查其中是否只包含一个@符号，以及是否有无效的字符（见图 6-12）。当然，脚本无法发现拼写错误，所以如果用户本打算输入 joe@myprovider.com，但是却输入了 joe@yprovider.com，脚本无法捕获这个错误。脚本 6-17 演示如何检查地址中的错误。



图 6-12　这是电子邮件验证脚本可以捕获的一种输入错误的例子

脚本 6-17　通过扫描表单上电子邮件字段中的文本，可以确保获得正确的电子邮件地址

```
window.onload = initForm;

function initForm() {
     document.forms[0].onsubmit = validForm;
     document.getElementById("sunroof").onclick = doorSet;
}

function validForm() {
     var allGood = true;
     var allTags = document.forms[0].getElementsByTagName("*");

     for (var i=0; i<allTags.length; i++) {
        if (!validTag(allTags[i])) {
           allGood = false;
        }
     }
     return allGood;
```

```
function validTag(thisTag) {
   var outClass = "";
   var allClasses = thisTag.className.split(" ");

   for (var j=0; j<allClasses.length; j++) {
      outClass += validBasedOnClass(allClasses[j]) + " ";
   }
   thisTag.className = outClass;

   if (outClass.indexOf("invalid") > -1) {
      invalidLabel(thisTag.parentNode);
      thisTag.focus();
      if (thisTag.nodeName == "INPUT") {
         thisTag.select();
      }
      return false;
   }
   return true;

   function validBasedOnClass(thisClass) {
      var classBack = "";

      switch(thisClass) {
         case "":
         case "invalid":
            break;
         case "reqd":
            if (allGood && thisTag.value == "") {
               classBack = "invalid ";
            }
            classBack += thisClass;
            break;
         case "radio":
            if (allGood && !radioPicked(thisTag.name)) {
               classBack = "invalid ";
            }
            classBack += thisClass;
            break;
         case "isNum":
            if (allGood && !isNum(thisTag.value)) {
               classBack = "invalid ";
            }
            classBack += thisClass;
            break;
         case "isZip":
            if (allGood && !isZip(thisTag.value)) {
               classBack = "invalid ";
            }
            classBack += thisClass;
            break;
         case "email":
            if (allGood && !validEmail(thisTag.value)) {
               classBack = "invalid ";
            }
            classBack += thisClass;
            break;
         default:
            if (allGood && !crossCheck(thisTag,thisClass)) {
               classBack = "invalid ";
            }
```

```
      classBack += thisClass;
   }
   return classBack;

function crossCheck(inTag,otherFieldID) {
   if (!document.getElementById(otherFieldID)) {
      return false;
   }
   return (inTag.value != "" || document.getElementById(otherFieldID).value != "");
}

function radioPicked(radioName) {
   var radioSet = document.forms[0][radioName];

   if (radioSet) {
      for (k=0; k<radioSet.length; k++) {
         if (radioSet[k].checked) {
            return true;
         }
      }
   }
   return false;
}

function isNum(passedVal) {
   if (passedVal == "") {
      return false;
   }
   for (var k=0; k<passedVal.length; k++) {
      if (passedVal.charAt(k) < "0") {
         return false;
      }
      if (passedVal.charAt(k) > "9") {
         return false;
      }
   }
   return true;
}

function isZip(inZip) {
   if (inZip == "") {
      return true;
   }
   return (isNum(inZip));
}

function validEmail(email) {
   var invalidChars = " /:,;";

   if (email == "") {
      return false;
   }
   for (var k=0; k<invalidChars.length; k++) {
      var badChar = invalidChars.charAt(k);
      if (email.indexOf(badChar) > -1) {
         return false;
      }
   }
   var atPos = email.indexOf("@",1);
   if (atPos == -1) {
```

6

```
                  return false;
               }
               if (email.indexOf("@",atPos+1)!= -1) {
                  return false;
               }
               var periodPos = email.indexOf(".",atPos);
               if (periodPos == -1) {
                  return false;
               }
               if (periodPos+3 > email.length) {
                  return false;
               }
               return true;
            }
         }
      }

      function invalidLabel(parentTag) {
         if (parentTag.nodeName == "LABEL") {
            parentTag.className += " invalid";
         }
      }
}

function doorSet() {
      if (this.checked) {
         document.getElementById("twoDoor").checked = true;
      }
}
```

⇒ 验证电子邮件地址

```
1. if (allGood && !validEmail(thisTag.value)){
     classBack = "invalid ";
```

这行代码添加在 switch/case 条件结构的 email 块中。如果 validEmail() 函数返回 false，就将 class 设置为 invalid。

```
2. var invalidChars = " /:,;";
```

在 validEmail 函数中，创建变量 invalidChars，其中包含电子邮件地址中最可能出现的 5 个无效字符：空格、斜杠、冒号、逗号和分号。

```
3. if (email == "") {
    return false;
```

这个测试的意思是，“如果 email 字段的内容是空的，那么结果是 false。”

```
4. for (var k=0; k<invalidChars.length; k++) {
```

在这个 for 语句中，对 invalidChars 字符串中的每个字符进行扫描。首先将计数器 k 初始化为零，然后设置循环终止条件（k 小于字符串的长度），最后用递增操作符++使 k 递增。

```
5. var badChar = invalidChars.charAt(k);
   if (email.indexOf(badChar) > -1) {
     return false;
```

将 badChar 变量设置为 invalidChars 字符串中 k 位置上的无效字符，然后检查电子邮件地址

（email）中是否有这个字符。如果有，indexOf()就会返回找到这个字符的位置；如果没有，它会返回–1。如果获得–1 之外的值，就说明找到了一个无效字符，因此返回 false。

6.
```
var atPos = email.indexOf("@",1);
 if (atPos == -1) {
   return false;
```

将 atPos 变量设置为@符号的位置。脚本使用 indexOf 从地址的第二个字符开始检查第一个@符号。如果这个@符号的位置是–1，就意味着地址中没有@符号，因此是错误的。

7.
```
if (email.indexOf("@",atPos+1) != -1) {
   return false;
```

现在，脚本检查第一个@符号之后的字符，从而确保只有一个@符号，并且拒绝任何包含多个@符号的输入。

6

8.
```
var periodPos = email.indexOf(".",atPos);
 if (periodPos == -1) {
   return false;
```

现在，脚本检查在@符号后面是否有点号。如果没有，就返回 false。

9.
```
if (periodPos+3 > email.length) {
   return false;
 }
 return true;
```

最后，脚本要求在地址中的点号后面至少有两个字符。如果通过了所有这些检查，而没有发现错误，那么 validEmail 函数的值就是 true，这意味着电子邮件地址是有效的。

✔提示

- 电子邮件地址验证和确认之间有一点区别。这个脚本只是验证电子邮件地址，确保用户输入的内容符合电子邮件地址的正确形式。但是，它无法确认这个地址是否真实存在。进行确认的唯一方法是向这个地址发送一封电子邮件，并检查是否有回复。但是，这种做法不太好，因为如果发送确认邮件，可能会引起用户的反感，而且邮件可能要过好几个小时才能得到回复，而在此期间用户不会有耐心一直在表单上等待。
- 这个脚本并未捕捉所有电子邮件地址错误，而是仅捕捉最常见的错误。对所有电子邮件地址错误进行全面检查需要好几页代码。如果你仔细研究一番，可能能够捕捉这个脚本未考虑到的错误。

第 7 章

表单和正则表达式

正则表达式是一种对文本字符串进行验证和格式化的极其强大的方式。通过使用正则表达式，可以用一两行 JavaScript 代码完成原本需要几十行代码的复杂任务。

正则表达式（regular expression，常常缩写为 RegExp）是一种用特殊符号编写的模式，描述一个或多个文本字符串。使用正则表达式匹配文本的模式，这样脚本就可以轻松地识别和操纵文本。与算术表达式一样，创建正则表达式也要使用操作符，但是在这种情况下使用的是操作文本（而不是数字）的操作符。有许多正则表达式操作符，我们在本章中将讨论最常用的一些操作符。通过学习和使用这些操作符，在需要搜索和操纵文本字符串时，就能够节省大量时间和精力。

正则表达式也常常被认为是编程中最棘手的部分之一。你可能觉得已经很好地掌握了 JavaScript，但是突然遇到了一个包含正则表达式的脚本，这可能使你完全无法理解脚本的作用。如果你不了解正则表达式的语法，那么根本猜不出会发生什么情况。“这一大堆乱七八糟的字符是什么意思？”

但是，只要把混乱的正则表达式分解成有意义的小块，其语法并不难理解。在本章中，我们将讲解正则表达式语法，并讨论如何使用正则表达式使代码更简洁、更强大。

你想暂时回避正则表达式吗

如果这是你第一次面对正则表达式，你现在很可能觉得压力很大。我们把这一章放在这里，是因为使用正则表达式验证表单输入是非常有意义的。但是本书中的其余内容并不依赖于本章，所以如果你想跳到下一章，直到具备更丰富的脚本编程经验之后再来学习本章，也是可以的。

另一方面，正则表达式是值得花时间学习的。正则表达式不仅在 JavaScript 中有用，在其他许多地方也可以使用正则表达式，例如其他编程语言（比如 Perl、Java、Python 和 PHP）、Apache 配置文件以及 BBEdit 和 TextMate 等文本编辑器。甚至 Adobe Dreamweaver 和 Microsoft Word（在一定程度上）使用正则表达式也可以实现更强大的搜索和替换。

7.1 用正则表达式验证电子邮件地址

在第 6 章中，有一个对电子邮件地址进行验证的示例。完成这个任务所需的脚本相当长。脚本 7-3 的作用在本质上与脚本 6-17 相同，但是通过使用正则表达式，脚本的代码少多了，而且可以得到更严格的结果。HTML 见脚本 7-1，CSS 见脚本 7-2。

脚本 7-1　电子邮件验证示例的 HTML

```
<!DOCTYPE html>
<html>
<head>
     <title>Email Validation</title>
     <link rel="stylesheet" href="script01.css">
     <script src="script01.js"></script>
</head>
<body>
<h2 class="centered">Email Validation</h2>
<form action="someAction.cgi">
     <p><label for="emailAddr">Email Address:<input id="emailAddr" type="text" size="30" class="email">
     →</label></p>
     <p><input type="submit" value="Submit"> <input type="reset"></p>
</form>
</body>
</html>
```

脚本 7-2　这段 CSS 是本章刚开始的几个任务所需要的

```
body {
     color: #000;
     background-color: #FFF;
}

input.invalid {
     background-color: #FF9;
     border: 2px red inset;
}

label.invalid {
     color: #F00;
     font-weight: bold;
}

.centered {
     text-align: center;
}
```

脚本 7-3　只需几行 JavaScript 代码就完成了验证电子邮件地址的复杂任务

```
window.onload = function() {
     document.forms[0].onsubmit = validForm;
}

function validForm() {
     var allGood = true;
     var allTags = document.forms[0].getElementsByTagName("*");

     for (var i=0; i<allTags.length; i++) {
        if (!validTag(allTags[i])) {
           allGood = false;
        }
     }
     return allGood;

     function validTag(thisTag) {
        var outClass = "";
        var allClasses = thisTag.className.split(" ");
```

7

```
        for (var j=0; j<allClasses.length; j++) {
           outClass += validBasedOnClass(allClasses[j]) + " ";
        }

        thisTag.className = outClass;

        if (outClass.indexOf("invalid") > -1) {
           invalidLabel(thisTag.parentNode);
           thisTag.focus();
           if (thisTag.nodeName == "INPUT") {
              thisTag.select();
           }
           return false;
        }
        return true;

        function validBasedOnClass(thisClass) {
           var classBack = "";

           switch(thisClass) {
              case "":
              case "invalid":
                 break;
              case "email":
                 if (allGood && !validEmail(thisTag.value)) {
                    classBack = "invalid";
                 }
              default:
                 classBack += thisClass;
           }
           return classBack;
        }

        function validEmail(email) {
           var re = /^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/;

           return re.test(email);
        }

        function invalidLabel(parentTag) {
           if (parentTag.nodeName == "LABEL") {
               parentTag.className += "invalid";
           }
        }
     }
  }
```

⇛ 使用正则表达式验证电子邮件地址

1. `var re = /^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/;`

天哪！这到底是什么？别担心，这只是 `validEmail()`函数中的一个正则表达式。我们将它分解开并逐段讲解。与任何 JavaScript 代码一样，正则表达式也要从左向右读。

首先，`re` 仅仅是一个变量。我们将它命名为 `re`，这样在以后使用它时就容易想起它是一个正则表达式。这行代码将 `re` 的值设置为等号右边的正则表达式。

正则表达式总是以斜杠（/）开头和结尾（当然，仍然有一个分号，表示 JavaScript 代码行结束，

但分号不是正则表达式的一部分）。斜杠之间的所有内容都是正则表达式的组成部分。

脱字符（^）表示我们要使用这个表达式检查以特定的字符串开头的字符串。如果去掉脱字符，那么即使字符串开头有一堆“垃圾字符”，电子邮件地址也可能被认为是有效的。

表达式\w 表示任意单一字符，包括 a ~ z、A ~ Z、0 ~ 9 和下划线。电子邮件地址必须以这些字符之一开头。

加号+表示我们要寻找前面条目的一次或多次出现。在这个示例中，电子邮件地址必须以字符 a ~ z、A ~ Z、0 ~ 9 或下划线的任意组合开头。

左圆括号(表示一个组。这意味着后面将要引用圆括号中的所有内容，所以现在将它们放在一个组中。

方括号[]用来表示可以出现在其中的任意一个字符。在这个示例中，方括号内包含字符\.-。我们希望允许用户输入点号或连字符，但是点号对于正则表达式有特殊意义，所以需要在它前面加上反斜杠\，这表示我们指的实际上是点号本身，而不是它的特殊意义。在特殊字符前面使用反斜杠称为“对字符转义”。因为有方括号，输入的字符串在这个位置可以有一个点号或一个连字符，但是两者不能同时存在。注意，连字符不代表任何特殊字符，所以不用加反斜杠。

问号?表示前面的条目可以不出现或者出现一次。所以，在电子邮件地址的第一部分（在@前面的部分）中可以有一个点号或一个连字符，也可以没有。

在?后面，再次使用\w+，这表示点号或连字符后面必须有其他一些字符。

右圆括号)表示这个组结束了。在此之后是一个星号，表示前面的条目（在这个示例中，指圆括号中的所有内容）可以不出现或者出现多次。所以如果 dori 是有效的电子邮件前缀，testing-testing-1-2-3 也是。

@字符仅仅代表它本身，没有任何其他意义，这个字符位于电子邮件地址前缀和域名之间。

再次使用\w+，这表示域名必须以一个或多个 a ~ z、A ~ Z、0 ~ 9 或下划线字符开头。在此之后同样是([\.-]?\w+)*，表示电子邮件地址的后缀中允许有点号或连字符。

然后，在一对圆括号中建立另一个组：\.\w{2,3}，表示我们希望找到一个点号，后面跟着一些字符。在这个示例中，花括号中的数字表示前面的条目（本例中是\w，表示字母、数字或下划线）可以出现 2 次或 3 次。在这个组的右圆括号后面是一个+，也表示前面的条目（这个组）必须出现一次或多次。这会匹配.com 或.edu 之类的，也与 ox.ac.uk 匹配。

最后，正则表达式的末尾是一个美元符号$，表示匹配的字符串必须在这里结束。这使脚本能够拒绝那些开头正确，但是在末尾包含垃圾字符的电子邮件地址。斜杠结束正则表达式。分号和原来一样结束 JavaScript 语句。

2. `return re.test(email);`

这一行获得前一步中定义的正则表达式，并使用 test()方法验证电子邮件地址的有效性。如果输入的字符串不符合 re 中存储的模式，test()就返回 false，错误的字段及其标签变成红色的粗体，如图 7-1 所示。如果输入有效，就返回 true（见图 7-2），表单将电子邮件地址提交给一个服务器端脚本（someAction.cgi）进行进一步处理。



图 7-1　这是用户输入无效电子邮件地址的结果，标签和字段变成红色、粗体



图 7-2　这个地址是有效的

✔提示

- 这段代码并没有匹配每一种合法的电子邮件地址形式，仅仅匹配你最想让用户输入的形式。
- 注意，在脚本 7-3 中为 re 赋值之后，在步骤 2 中将 re 作为对象使用。与其他 JavaScript 变量一样，正则表达式的结果可以是一个对象。
- 请比较脚本 6-17 和脚本 7-3 中的 validEmail()函数。前者有 27 行，而后者只有 4 行。它们的作用是相同的，由此可以看出正则表达式确实可以减少大量代码。
- 在表 7-1 中会看到正则表达式中的特殊字符（有时候称为元字符，meta character）是区分大小写的。在对使用正则表达式的脚本进行调试时要记住这一点。
- 在正则表达式中，有一些字符可以改变其他操作符的行为，见表 7-2。

表7-1　正则表达式中的特殊字符

字　符	匹　配
\	在字面意义和特殊意义之间进行切换。例如\w表示\w的特殊意义（见下面的解释）而不是字面值w，但是\$表示不使用$的特殊意义（见下面的解释）而是使用$字符本身
^	字符串的开头
$	字符串的结尾
*	零次或多次
+	一次或多次
?	零次或一次
.	除换行符外的任何字符
\b	单词边界
\B	非单词边界
\d	0～9的任何数字（与[0-9]相同）
\D	任何非数字
\f	换页符（form feed）
\n	换行符
\r	回车符
\s	任何一个空白字符（与[\f\n\r\t\v]相同）
\S	任何一个非空白字符
\t	制表符
\v	垂直制表符

（续）

字　符	匹　配
\w	任何字母、数字以及下划线（与[a-zA-Z0-9_]相同）
\W	除数字、字母及下划线外的其他字符
\xnn	十六进制数字nn定义的ASCII字符
\onn	八进制数字nn定义的ASCII字符
\cX	控制字符X
[abcde]	与其中任何字符匹配的字符集
[^abcde]	字符补集，与其中任何字符都不匹配的字符集
[a-e]	与其中的字符范围匹配的字符集
[\b]	退格字符的字面意义（不同于\b）
{n}	前面的字符正好出现*n*次
{n,}	前面的字符至少出现*n*次
{n,m}	前面的字符出现*n*～*m*次
()	一个组，可以在后面引用它
x|y	x或y

表7-2　正则表达式修饰符

修饰符	含　义
g	搜索所有的匹配（全局），不只是第一处匹配
i	进行不区分大小写的搜索

7.2　验证文件名

可以用正则表达式做许多事情，但是最有用的功能之一是验证网页上表单中的输入字段。脚本 7-4 希望用户输入一个图像的有效 URL，正则表达式有助于确保用户的输入符合要求（具体地说，文件名必须有表示图像文件的后缀）。图 7-3 显示当意外输入无效的 URL 时页面的效果，图 7-4 显示输入正确的图像名称时的效果。



图 7-3　由于使用了正则表达式，如果用户输入的不是有效的图像文件名，页面就会指出这个错误

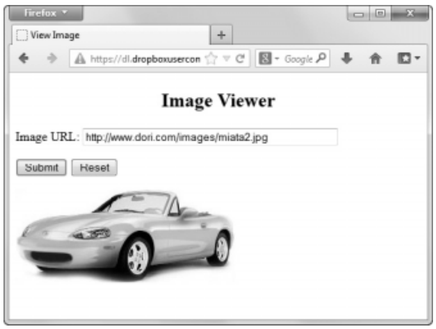


图 7-4　如果输入了正确的图像文件名，就在页面上显示这个图像

脚本 7-4　这个脚本要求用户输入图像位置，如果 URL 通过了验证，就在页面上显示这个图像

```
window.onload = function() {
     document.forms[0].onsubmit = validForm;
}

function validForm() {
     var allGood = true;
     var allTags = document.forms[0].getElementsByTagName("*");

     for (var i=0; i<allTags.length; i++) {
        if (!validTag(allTags[i])) {
           allGood = false;
        }
     }
     return false;

     function validTag(thisTag) {
        var outClass = "";
        var allClasses = thisTag.className.split(" ");

        for (var j=0; j<allClasses.length; j++) {
           outClass += validBasedOnClass(allClasses[j]) + " ";
        }

        thisTag.className = outClass;

        if (outClass.indexOf("invalid") > -1) {
           invalidLabel(thisTag.parentNode);
           thisTag.focus();
           if (thisTag.nodeName == "INPUT") {
              thisTag.select();
           }
           return false;
        }
        return true;

        function validBasedOnClass(thisClass) {
           var classBack = "";

           switch(thisClass) {
              case "":
              case "invalid":
                 break;
              case "imgURL":
                 if (allGood && !setImgURL(thisTag.value)) {
                    classBack = "invalid ";
                 }
              default:
                 classBack += thisClass;
           }
           return classBack;
        }

        function setImgURL(newURL) {
           var re = /^(file|http):\/\/\S+\/\S+\.(gif|jpg|png)$/i;

           if (re.test(newURL)) {
              document.getElementById("chgImg").src = newURL;
              return true;
```

```
            }
            return false;
        }

        function invalidLabel(parentTag) {
            if (parentTag.nodeName == "LABEL") {
                parentTag.className += "invalid";
            }
        }
    }
}
```

⇛ 验证 URL

- var re = /^(file|http):\/\/\S+\/\S+\.(gif|jpg/png)$/i;

这行代码在 imgURL()函数中。与前一个示例一样，我们希望检查输入的整个字段，所以正则表达式以/^开头，以$/结束。输入可以以文本 http 或 file 开头，所以将这两个字符串放在一个组中，用|分隔，表示可以接受两者之一。无论用户是从本地硬盘还是从 Web 获得图像，://几个字符都是必需的，所以接下来检查这些字符。注意，每个正斜杠必须分别进行转义（这里的两处\/都是转义后的正斜杠），因为正斜杠是正则表达式特殊字符。

在此之后，几乎可以出现任何字符，所以使用\S+表示后面是一个或多个非空格字符。然后需要另一个正斜杠（同样经过转义）来分隔域名和文件名，然后是另一个\S+用来处理文件名。

文件名需要以点号和 gif、jpg 和 png 结束。点号经过转义，两个后缀组合在一起，表示接受其中之一。

在这个正则表达式后面，使用修饰符 i 允许用户输入大写或小写字母。这个修饰符让正则表达式不区分大小写。

7.3 提取字符串

字符串验证并不是使用正则表达式能够完成的唯一任务。字符串提取（extraction）也是有意义的。可以提取并操作字符串的一部分，这样就可以对最终结果进行更细的控制。在脚本 7-5 中，我们获得一系列输入的姓名，其中名字在前，姓氏在后，然后交换名和姓的次序。

脚本 7-5 这个脚本重新安排输入的姓名列表

```
window.onload = function() {
     document.forms[0].onsubmit = validForm;
}

function validForm() {
     var allTags = document.forms[0].getElementsByTagName("*");

     for (var i=0; i<allTags.length; i++) {
        validTag(allTags[i]);
     }
     return false;

     function validTag(thisTag) {
        var allClasses = thisTag.className.split(" ");

        for (var j=0; j<allClasses.length; j++) {
           if (allClasses[j] == "nameList") {
```

7

```
                thisTag.value = setNameList(thisTag.value);
            }
        }

        function setNameList(inNameList) {
            var newNames = new Array;
            var newNameField = "";

            var re = /\s*\n\s*/;
            var nameList = inNameList.split(re);

            re = /(\S+)\s(\S+)/;

            for (var k=0; k<nameList.length; k++) {
                newNames[k] = nameList[k].replace(re, "$2, $1");
            }

            for (k=0; k<newNames.length; k++) {
                newNameField += newNames[k] + "\n";
            }
            return newNameField;
        }
    }
}
```

⇛ 提取字符串

1. `var re = /\s*\n\s*/;`

这是一个新的正则表达式，它搜索的文本模式是按照任何空白字符（`\s*`）、换行符（`\n`）和任何空白字符（`\s*`）的顺序组成的。

2. `var nameList = inNameList.split(re);`

字符串方法 `split()`获得正则表达式，并且将它应用于用户输入的存储在 `inNameList` 中的数据（见图 7-5）。每个换行符分隔一个姓名，`split()`将每行上的输入数据分隔开。结果是一个输入的姓名的字符串数组，其中每个数组元素都是一个姓名，存储在数组 `nameList` 中。

3. `re = /(\S+)\s(\S+)/;`

接下来，需要另一个正则表达式，它可以将每个姓名分隔成名字和姓氏。它搜索的是任何非空白字符（`\S+`），然后是一个空白字符（`\s`），最后是任何非空白字符（`\S+`）。每组非空白字符需要加上圆括号，以便在后面引用这些字符。

4. `for (var k=0; k<nameList.length; k++) {`

对于 `nameList` 数组中的每个姓名，循环执行以下代码。

5. `newNames[k] = nameList[k].replace (re, "$2, $1");`

记得步骤 3 中的圆括号吗？当执行 `replace()`方法时，正则表达式 `re` 将 `nameList` 的数组元素分隔成名字和姓氏。这些圆括号让 JavaScript 将名字存储在正则表达式属性`$1` 中，将姓氏存储在正则表达式属性`$2` 中。然后，`replace()`方法使用传递给它的第二个参数，返回新的字符串：首先是姓氏`$2`，然后是一个逗号，最后是名字`$1`。现在，将姓氏在前的姓名存储在新数组 `newNames` 中。

6.
```
for (k=0; k<newNames.length; k++) {
    newNameField += newNames[k] + "\n";
}
```

这个循环设置一个新变量 newNameField，它将包含转换后的用户输入文本。对于 newNames 数组中的每个姓名，将姓名追加进 newNameField，再加上一个换行符。

7. return newNameField;

我们返回结果来更新网页。更新在 switch/case 部分：thisTag.value = nameList(thisTag.value);。结果见图 7-6。

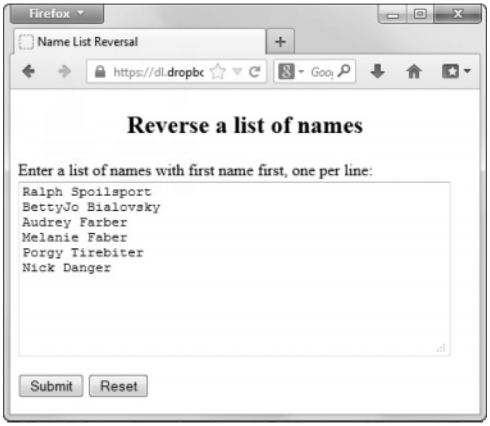


图 7-5　这是转换前的列表

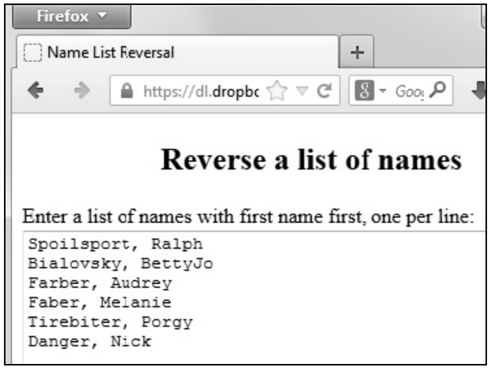


图 7-6　改变姓名次序后的页面

✔提示

- 这个脚本只处理由空格分隔的名字和姓氏。如果希望处理中间名或者由多部分组成的姓氏，就必须修改脚本。
- 在这个脚本中，在不同的位置多次使用了变量 re，并给它赋以不同的值。这在 JavaScript 中是允许的（我们在这里这么做就是为了演示这一点），但是也可以考虑在自己的脚本中使用不同的变量名。在以后调试或修改脚本时，这会使工作更轻松。

7.4　格式化字符串

用户常常以很随意的格式输入数据。如果你希望输入符合一种标准格式，那么最好自己处理格式化。脚本 7-6 演示了如何获得一系列姓名并且将它们转换为标准的首字母大写格式。

脚本 7-6　这个脚本获得以任何格式输入的姓名，并且将它们转换为首字母大写格式

```
window.onload = function() {
     document.forms[0].onsubmit = validForm;
}

function validForm() {
     var allTags = document.forms[0].getElementsByTagName("*");

     for (var i=0; i<allTags.length; i++) {
        validTag(allTags[i]);
     }
     return false;
```

```
    function validTag(thisTag) {
       var allClasses = thisTag.className.split(" ");

       for (var j=0; j<allClasses.length; j++) {
          if (allClasses[j] == "nameList") {
             thisTag.value = setNameList(thisTag.value);
          }
       }

       function setNameList(inNameList) {
          var newNames = new Array;
          var newNameField = "";

          var re = /\s*\n\s*/;
          var nameList = inNameList.split(re);

          re = /^(\S)(\S+)\s(\S)(\S+)$/;

          for (var k=0; k<nameList.length; k++) {
             if (nameList[k]) {
                re.exec(nameList[k]);
                newNames[k] = RegExp.$1.toUpperCase() + RegExp.$2.toLowerCase() + " " + RegExp.$3.
                ➝toUpperCase() + RegExp.$4.toLowerCase();
             }
          }
          for (k=0; k<newNames.length; k++) {
             newNameField += newNames[k] + "\n";
          }
          return newNameField;
       }
    }
}
```

⇛ 对字符串进行格式化

1. `re = /^(\S)(\S+)\s(\S)(\S+)$/;`

这个正则表达式同样寻找符合“名字、空格、姓氏”次序的姓名，并且将每个姓名分隔成 4 部分：名字的首字母`^(\S)`、名字的剩余字母`(\S+)`、姓氏的首字母`(\S)`以及姓氏的剩余字母`(\S+)$`。注意，`^`和`$`迫使字符串在这两个位置开始和结束，因为我们不希望漏掉任何东西。

2. `for (var k=0; k<nameList.length; k++) {`

我们希望检查 `nameList` 数组中的每个姓名，见图 7-7。

3. `re.exec(nameList[k]);`

这个步骤使用 `exec()`方法在字符串 `nameList[k]`上执行正则表达式 `re`，从而将字符串分隔为 4 个部分，并且自动地设置 JavaScript 内置的 `RegExp` 对象。这 4 个部分分别存储在 `RegExp.$1`、`RegExp.$2`、`RegExp.$3` 和 `RegExp.$4` 中。

4. `newNames[k] = RegExp.$1.toUpperCase()+ RegExp.$2.toLowerCase() +" " + RegExp.$3.`
   `➝toUpperCase()+ RegExp.$4.toLowerCase();`

转换后的姓名存储在 `newNames` 数组中。它包含转换为大写的名字首字母（`RegExp.$1`），转换为小写的名字剩余字母（`RegExp.$2`），然后是一个空格，然后是转换为大写的姓氏首字母（`RegExp.$3`），最后是转换为小写的姓氏剩余字母（`RegExp.$4`）。然后显示姓名，见图 7-8。



图 7-7　这是转换之前的姓名



图 7-8　这是转换后的效果，已经符合我们的要求了

关于 RegExp 对象

JavaScript 有一个内置的 RegExp 对象，每当脚本执行正则表达式方法（见表 7-4 和表 7-5）时，会自动地设置（和重新设置）这个对象。这个对象的属性见表 7-3，它的方法见表 7-4。RegExp 对象并不是一个包含正则表达式操作结果的变量，而是包含正则表达式所描述的模式，脚本可以通过 RegExp 对象的属性和方法访问文本模式的各个部分。

表7-3　RegExp对象的属性

属　　性	意　　义
$1（到$9）	圆括号包围的子字符串匹配
$_	相当于input
$*	相当于multiline
$&	相当于lastMatch
$+	相当于lastParen
$`	相当于leftContext
$'	相当于rightContext
constructor	指定创建对象原型的函数
global	全局搜索（使用g修饰符）
ignoreCase	不区分大小写搜索（使用i修饰符）
input	如果没有传递字符串，这就是要搜索的字符串
lastIndex	继续匹配的起始位置
lastMatch	最后一个匹配的字符串
lastParen	最后的圆括号包围的子字符串匹配
leftContext	最近一个匹配字符串左边的子字符串
multiline	是否跨多行搜索字符串
prototype	允许在所有对象中添加属性
rightContext	最近一个匹配字符串右边的子字符串
source	正则表达式模式本身

表7-4　RegExp对象的方法

方　　法	意　　义
compile(pattern,[, "g" | "i" | "gi"])	对正则表达式进行编译
exec(string)	搜索匹配
test(string)	测试匹配
toSource()	返回一个代表对象的字面值
toString()	返回一个代表指定对象的字符串
valueOf()	返回指定对象的原始值

表7-5　字符串方法

方　　法	意　　义
match(re)	在一个字符串中寻找与一个正则表达式模式（re）的匹配
replace(re, replaceStr)	使用正则表达式（re）执行所需的替换
search(re)	搜索与正则表达式（re）的匹配
split(re)	根据正则表达式（re）对字符串进行分隔

7.5　对字符串进行格式化和排序

另一种典型任务是对一组姓名进行排序。脚本 7-7 将前两个示例组合起来并且添加了排序功能。最终结果是一个姓氏在前的姓名列表，采用首字母大写形式，并且按照字母表排序。

脚本 7-7　这个脚本接受一系列任意格式和次序的姓名，并且将它们转换成标准格式的排序列表

```
window.onload = function() {
     document.forms[0].onsubmit = validForm;
}

function validForm() {
     var allTags = document.forms[0].getElementsByTagName("*");

     for (var i=0; i<allTags.length; i++) {
        validTag(allTags[i]);
     }
     return false;

     function validTag(thisTag) {
        var allClasses = thisTag.className.split(" ");

        for (var j=0; j<allClasses.length; j++) {
           if (allClasses[j] == "nameList") {
              thisTag.value = setNameList(thisTag.value);
           }
        }

        function setNameList(inNameList) {
           var newNames = new Array;
           var newNameField = "";
```

```
            var re = /\s*\n\s*/;
            var nameList = inNameList.split(re);

            re = /^(\S)(\S+)\s(\S)(\S+)$/;

            for (var k=0; k<nameList.length; k++) {
               if (nameList[k]) {
                  re.exec(nameList[k]);
                  newNames[k] = RegExp.$3.toUpperCase() + RegExp.$4.toLowerCase() + ", " + RegExp.$1.
                  ➝toUpperCase() + RegExp.$2.toLowerCase();
               }
            }

            newNames.sort();
            for (k=0; k<newNames.length; k++) {
               newNameField += newNames[k] + "\n";
            }
            return newNameField;
         }
      }
}
```


⇛ 对字符串进行格式化和排序

1. `newNames[k] = RegExp.$3.toUpperCase() + RegExp.$4.toLowerCase() + ", " + RegExp.$1.`
   `➝toUpperCase() + RegExp.$2.toLowerCase();`

在这个示例中，我们希望按照姓氏进行排序，所以创建一个新的 newNames 数组，其元素的构成是转换为大写的姓氏首字母，转换为小写的姓氏剩余字母，然后是一个逗号和一个空格，再是转换为大写的名字首字母，最后是转换为小写的名字剩余字母。

2. `newNames.sort();`

数组方法 sort()对数组中的元素进行排序，并覆盖原来的内容。图 7-9 显示转换之前的版本，图 7-10 显示转换之后的版本。



图 7-9　这是用户输入的姓名列表



图 7-10　这是排序和整理之后的列表，符合了我们的要求

7.6　对字符串进行格式化和验证

可以使用正则表达式对所输入的值同时进行格式化和验证。在脚本 7-8 中，用户输入一个任意格

式的电话号码。最终结果要么是一个经过格式化的电话号码，要么是输入框变成红色，标签变成红色、粗体。

脚本 7-8　这个脚本对用户输入的电话号码进行验证和格式化

```
window.onload = function() {
     document.forms[0].onsubmit = validForm;
}

function validForm() {
     var allTags = document.forms[0].getElementsByTagName("*");

     for (var i=0; i<allTags.length; i++) {
        validTag(allTags[i]);
     }
     return false;

     function validTag(thisTag) {
        var outClass = "";
        var allClasses = thisTag.className.split(" ");

        for (var j=0; j<allClasses.length; j++) {
           outClass += validBasedOnClass(allClasses[j]) + " ";
        }

        thisTag.className = outClass;

        if (outClass.indexOf("invalid") > -1) {
           invalidLabel(thisTag.parentNode);
           thisTag.focus();
           if (thisTag.nodeName == "INPUT") {
              thisTag.select();
           }
        }
        function validBasedOnClass(thisClass) {
           var classBack = "";

           switch(thisClass) {
              case "":
              case "invalid":
                 break;
              case "phone":
                 if (!validPhone(thisTag.value)) {
                    classBack = "invalid ";
                 }
              default:
                 classBack += thisClass;
           }
           return classBack;
        }

        function validPhone(phoneNum) {
           var re = /^\(?(\d{3})\)?[\.\-\/]?(\d{3})[\.\-\/ ]?(\d{4})$/;

           var phoneArray = re.exec(phoneNum);
           if (phoneArray) {
              document.getElementById("phoneField").value = "(" + phoneArray[1] + ") " + phoneArray[2]
              ➝+ "-" + phoneArray[3];
              return true;
```

```
            }
            return false;
         }

         function invalidLabel(parentTag) {
            if (parentTag.nodeName == "LABEL") {
               parentTag.className += "invalid";
            }
         }
      }
   }
```

⇛ 对电话号码进行格式化和验证

1. `var re = /^\(?(\d{3})\)?[\.\-\/ ]?(\d{3})[\.\-\/ ]?(\d{4})$/;`

这个正则表达式寻找这样的字符串：

- 有一个可选的左圆括号`\(?`
- 有 3 个数字`(\d{3})`
- 有一个可选的右圆括号`\)?`
- 有一个可选的点号、连字符、正斜杠或空格`[\.\-\/ ]?`
- 有 3 个数字`(\d{3})`
- 有一个可选的点号、连字符、正斜杠或空格`[\.\-\/ ]?`
- 有 4 个数字`(\d{4})`

这个模式指定了字符串的开头和结尾，所以如果有额外的字符，字符串就不匹配。如果找到三位的地区编码数字、三位的前缀数字和四位的后缀数字，就分别保存它们。

2. `var phoneArray = re.exec(phoneNum);`

`exec()`方法在 `phoneNum` 上执行 `re` 中存储的正则表达式。如果没有找到要搜索的模式（见图 7-11），`phoneArray` 就被设置为 `null`。否则，`phoneArray` 就设置为一个数组，其中包含正则表达式存储的值。

3.
```
if (phoneArray) {
   document.getElementById("phoneField").value = "(" + phoneArray[1] + ") "+
     phoneArray[2] + "-" + phoneArray[3];
```

如果 `phoneArray` 非空，就会成功地通过这个测试，这说明数组已经初始化了。所以，将页面上的表单字段重新设置为标准格式：包围在圆括号中的地区编码，加一个空格，然后是前缀、连字符和后缀，如图 7-12 所示。



图 7-11　这是输入无效号码时的结果



图 7-12　这是输入正确的号码时的结果

7.7　使用正则表达式替换元素

你已经看到了正则表达式在搜索、匹配和替换字符串方面是多么强大。但是，还可以使用它们替换页面元素的名称，这常常可以节省大量开发时间。在本节中，我们用一个正则表达式改造前面的脚本 4-6。这个脚本建立了一个三状态翻转器。它是一个有用的脚本，但是有一个缺点：它要求给希望操纵的每个图像加上它自己的 id。这并不太困难，但是可以使用 JavaScript 构建页面元素的名称，从而减少工作量。

现在，你应该回顾一下脚本 4-5 和脚本 4-6 的工作方式。稍等片刻，我们继续前进。

脚本 4-6 原来根据每个图像的 id 动态地创建图像的_click 和_on 名称。现在不这样做，而是根据每个图像的_off 名称动态地创建_click 和_on 名称。这样就不需要图像 id 了。脚本 7-9 演示了实现方法。修改 JavaScript 并没有改变这个页面的外观或行为，但是创建 HTML 页面所需的工作量减少了。

脚本 7-9　可以使用正则表达式减少编写或修改 HTML 文件的工作量

```
window.onload = rolloverInit;

function rolloverInit() {
     for (var i=0; i<document.images.length;i++) {
        if (document.images[i].parentNode.tagName.toLowerCase() == "a") {
           setupRollover(document.images[i]);
        }
     }
}

function setupRollover(theImage) {
     var re = /\s*_off\s*/;

     theImage.outImage = new Image();
     theImage.outImage.src = theImage.src;
     theImage.onmouseout = function() {
        this.src = this.outImage.src;
     }

     theImage.overImage = new Image();
     theImage.overImage.src = theImage.src.replace(re,"_on");
     theImage.onmouseover = function() {
        this.src = this.overImage.src;
     }

     theImage.clickImage = new Image();
     theImage.clickImage.src = theImage.src.replace(re,"_click");
     theImage.onclick = function() {
        this.src = this.clickImage.src;
     }

     theImage.parentNode.childImg = theImage;

     theImage.parentNode.onblur = function() {
        this.childImg.src = this.childImg.outImage.src;
     }

     theImage.parentNode.onfocus = function() {
        this.childImg.src = this.childImg.overImage.src;
     }
}
```

⇛ 使用正则表达式替换元素

1. `var re = /\s*_off\s*/;`

这一行设置一个新的正则表达式模式，它在字符串中的任何地方寻找文本_off。

2. `theImage.overImage.src = theImage.src.replace(re,"_on");`

脚本 4-6 中的这一行是 `theImage.overImage.src ="images/"+ theImage.id + "_on.gif";`。新的代码使用 re 模式查找特殊的字符串，如果找到，就替换它。本例中我们搜索字符串中的_off，并把它替换为_on。这样我们就不必为在图像上设置 id 属性费心了，根本不需要 id 了。

3. `theImage.clickImage.src = theImage.src.replace(re,"_click");`

脚本 4-6 中的这一行是 `theImage.clickImage.src ="images/"+ theImage.id + "_click.gif";`。新的代码搜索_off 并且将它替换为_click。

✔提示

- ❑ 如果你的图像采用.gif、.jpg 和.png 三种格式，这个脚本也会很方便。现在的 JavaScript 代码甚至不需要知道每个图像的扩展名是什么。
- ❑ 你可能已经注意到，这个脚本末尾的一些代码是脚本 4-6 中没有的。在这里添加这些代码是为了提高可访问性。现在，那些使用键盘而不用鼠标的用户可以通过制表键移动到图像上，这会产生与把鼠标移动到图像上相同的效果。

第 8 章

处理事件

8

事件是用户在访问页面时执行的操作。当浏览器探测到一个事件时，比如用鼠标单击或按键，它可以触发与这个事件相关联的 JavaScript 对象，这些对象称为事件处理程序（event handler）。在本书前面的许多地方，你已经见到了使用事件处理程序的示例。但是，事件处理是一项重要的技术，而且它实际上包含了用户与页面的所有交互，所以值得用单独的一章来讲解它。

在本章中，你将看到如何使用事件处理程序来操作窗口、捕获鼠标移动和单击、处理表单事件以及在用户按下键盘上的键时作出响应。

8.1 处理窗口事件

当用户执行某些会影响整个浏览器窗口的操作时，就会发生窗口事件。最常见的窗口事件是通过打开某个网页来加载窗口。还有在窗口关闭、移动或转到后台时触发事件处理程序的事件。

在使用事件处理程序时，常常会发现使用点号语法将事件处理程序与一个对象连接起来是有意义的，如下所示：

```
window.onfocus
window.onload
document.onmousedown
```

注意，像这样将事件处理程序作为对象的一部分使用时，事件处理程序的名称是全小写的。另外，更符合标准的做法是，将事件处理程序放在外部脚本而不是 HTML 标签中，这样可以将 JavaScript 代码与 HTML 代码分隔开，而且在一个外部文件中编辑（或替换）所有 JavaScript 代码会更容易。

8.1.1 onload事件

在本书中，我们经常使用 `onload` 事件。当用户进入页面而且所有页面元素都完成加载时，就会触发这个事件。流行的广告弹出窗口就是使用 `onload` 事件处理程序的典型例子，尽管这种方式可能让用户不快。

尽管我们已经多次在脚本中见过使用 `onload` 的情况，但是到目前为止，我们一直回避了一个重要的问题：如果在加载页面时需要进行多个操作，那么应该怎么做？脚本 8-1 和脚本 8-2 演示了具体做法。

脚本 8-1 多重 onload 示例的 HTML

```
<!DOCTYPE html>
<html>
<head>
     <title>Welcome!</title>
     <script src="script01.js"></script>
</head>
<body id="pageBody">
     <h1>Welcome to our Web site!</h1>
</body>
</html>
```

脚本 8-2 使用新的 addOnload()函数设置多重 onload 属性

```
addOnload(initOne);
addOnload(initTwo);
addOnload(initThree);

function addOnload(newFunction) {
     var oldOnload = window.onload;

     if (typeof oldOnload == "function") {
        window.onload = function() {
           oldOnload();
           newFunction();
        }
     }
     else {
        window.onload = newFunction;
     }
}

function initOne() {
     document.getElementById("pageBody").style.backgroundColor = "#00F";
}

function initTwo() {
     document.getElementById("pageBody").style.color = "#F00";
}

function initThree() {
     var allTags = document.getElementById("PageBoby")getElementsByTagName("*");

     for (var i=0; i<allTags.length; i++) {
        if (allTags[i].nodeName == "H1") {
           allTags[i].style.border = "5px green solid";
           allTags[i].style.padding = "25px";
           allTags[i].style.backgroundColor = "#FFF";
        }
     }
}
```

1. `addOnload(initOne);`
   `addOnload(initTwo);`
   `addOnload(initThree);`

在这个脚本中，在首次加载页面时，我们希望发生完全不同的三种情况。设置 window.onload 三次是不行的，因为第二次设置会覆盖第一次的，第三次设置会覆盖第二次的。相反，我们要调用一个

8

新函数 addOnload()，由它替我们处理 onload 处理程序。对于每个调用，传递一个参数：在触发 onload 事件时希望运行的函数的名称。结果见图 8-1。



图 8-1　这个脚本设置在加载页面时运行的多个 onload 处理程序（在这个示例中，用于进行颜色格式化）

2. function addOnload(newFunction) {

与其他函数一样，这一行开始一个新函数。传递给它的参数是一个函数名。

这可能让你觉得有点儿困惑，所以下面举一个例子。我们并不调用

```
window.onload = myNewFunction;
```

而是调用

```
addOnload(myNewFunction);
```

其最终效果是相同的。

3. var oldOnload = window.onload;

这一行声明一个新变量 oldOnload。如果已经设置了 window.onload，就将它的值存储在这里。如果还没有设置，这一行也没什么坏处。

4. if (typeof oldOnload == "function") {

我们在这一行中检查 oldOnload 变量的类型。如果已经设置了 window.onload，那么它应该是一个函数调用（否则，就是空的）。如果它是函数，就执行以下代码。

```
5. window.onload = function() {
     oldOnload();
     newFunction();
   }
```

这些代码重新设置 window.onload 的值来完成两种操作：它之前所做的操作以及参数中传递的新函数。window.onload 事件处理程序设置为一个匿名函数（没有函数名的函数）。然后执行 window.onload 原本应该完成的操作。但是，在函数结束之前，还要执行 newFunction()。

```
6. else {
     window.onload = newFunction;
   }
```

如果 oldOnload 不是函数（也就是说，它是未定义的），就在页面完成加载时执行新函数。按照这种方式，我们可以多次调用 addOnload()：第一次它将自己的函数赋值给 window.onload；第二次和第三次会创建匿名函数，让 JavaScript 执行以前设置的操作和新添加的函数。

✔提示

- 如果想让一个 onload 处理程序执行多个操作，最简单的方法是创建一个执行所有操作的函数，然后让 onload 处理程序调用这个函数。但是，要确保每个函数都返回。例如，如果你的函数包含对本身的 setTimeout()调用，它将不会返回，因此不会到达被调用函数的其余部分。
- 如果你正在修改现有的代码，那么很容易意外地重新设置 window.onload——任何 HTML 页面都可以调用多个外部 JavaScript 文件，这些文件都可以设置事件处理程序。如果其中一个文件直接设置了 window.onload，但是每次你都在此之后调用 addOnload()，那么不会有问题。但是，如果在设置 window.onload 之后（无论是直接设置，还是通过 addOnload()），再设置 window.onload，原来设置的函数就会丢失。
- 这个脚本在一定程度上基于 Simon Willison（simonwillison.net）的一个脚本，在本书中使用它已经得到了 Simon 的许可。
- 8.5 节介绍了另外一种一个事件处理程序处理多个事件的方法。

8.1.2 onunload事件

当用户离开网页时，就会触发 onunload 事件处理程序。这个事件最常见的用途是，当用户离开某些商业站点（尤其是色情站点）时弹出广告窗口。如果你访问色情站点的话，常常会发现几乎不可能离开——每当你试图关闭窗口或导航到别处时，都会出现一个接一个的窗口，重新打开同样的页面或同类的其他页面。因此，用户非常讨厌 onunload 处理程序，所以使用它时要慎重。

但是，这不代表 onunload 处理程序无用武之地，6.1 节的脚本 6-3 就展示了 onunload 的一个不错的用法。

8.1.3 onbeforeunload事件

乍一看，onbeforeunload 的作用似乎和 onunload 一样，但两者有个很大的区别：onbeforeunload 在用户*开始*离开页面之前触发，而 onunload 在用户离开页面*之后*触发。脚本 8-3 和脚本 8-4 展示了两者的不同。

脚本 8-3　这个表单很简单，我们想确保用户不会因为意外情况离开表单

```
<!DOCTYPE html>
<html>
<head>
     <title>FabulousAirTickets.com</title>
     <script src="script02.js"></script>
</head>
<body>
<h2>FabulousAirTickets.com</h2>
<form action="#">
     <p>
        <label for="from">From:</label>
        <input type="text" id="from">
          
        <label for="to">To:</label>
        <input type="text" id="to">
     </p>
```

```
        <p>
           <label for="leavedate">DepartureDate:</label>
           <input type="date" id="leavedate">
             
           <label for="returndate">ReturnDate:</label>
           <input type="date" id="returndate">
        </p>
        <p>
           # of Adults:
           <select>
              <option value="1">1</option>
              <option value="2">2</option>
              <option value="3">3</option>
              <option value="4">4</option>
           </select>
        </p>
        <p>
           <input type="submit" value="Find Flights!">
             
           <input type="reset">
        </p>
   </form>
   </body>
   </html>
```

脚本 8-4　当用户尝试离开页面的时候，脚本会警告用户将丢失数据

```
window.onbeforeunload = function() {
     return "If you close this window, your flight choices will be lost!";
}
```

❑ window.onbeforeunload = function() {

  return "If you close this window, your flight choices will be lost!";

}

此处，我们想警告用户：如果他们离开表单，之前输入的所有信息都会丢失。如果用 onunload 来实现这个功能，用户之前输入的信息就全没了；但是因为这里使用的是 onbeforeunload，用户就有机会决定是否放弃之前输入的信息。

函数需要返回确认对话框中呈现给用户的信息，如图 8-2 所示。不同浏览器确认对话框中的信息不同，实际上，Firefox 压根就不用函数返回的信息。

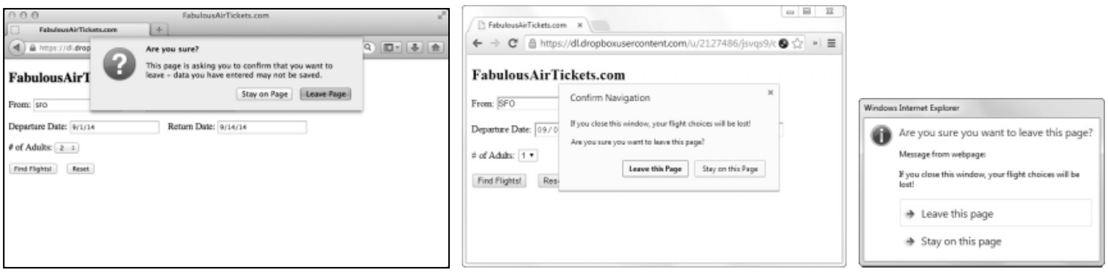


图 8-2　确保用户知道他们正在进行什么操作。不同浏览器的提示信息不同（从左到右依次为 Firefox、Chrome 和 IE）

8.1.4　onresize事件

当窗口调整大小时，会触发 onresize 事件处理程序。

8.1.5　onmove事件

当窗口移动时，会触发 onmove 事件处理程序。

8.1.6　onabort事件

当用户取消网页上的图像加载时，会触发 onabort 事件处理程序。这种事件不太常用，而且并非所有浏览器都支持它。

8.1.7　onerror事件

当页面上发生 JavaScript 错误时，可能会触发 onerror 事件处理程序。

✔提示

- 在 Web 上的复杂页面中，设置 onerror = null 会比较好。如果在页面上使用这行代码，某些错误消息将不会向用户显示，这样用户就会少受干扰，但是，究竟隐藏哪些错误取决于浏览器。

8

8.1.8　onfocus事件和onblur事件

onfocus 和 onblur 事件处理程序互为镜像，听起来就像现实中你在 JavaScript 程序上奋战到午夜一样。当一个页面成为最前面的活动窗口时，就会触发 onfocus 处理程序；而当一个页面退回后台的时候，就会触发 onblur 事件处理程序。

由于这两个事件处理程序老是被滥用，大多数现代浏览器已经不支持它们了。

8.1.9　onscroll事件

当用户向上或者向下滚动页面时，就会触发 onscroll 事件。

8.1.10　onDOMContentLoaded事件

onDOMContentLoaded 跟 onload 类似，只是它是在页面自身完成加载时被触发，而不是一些相关文件（如图片）完成加载时被触发。

8.2　处理鼠标事件

用户与页面的许多交互都是通过鼠标移动或鼠标单击进行的。JavaScript 为这些事件提供了一组强健的处理程序。

8.2.1 onmousedown事件

JavaScript 新手最经常问到的问题之一是，“如何对访问我的页面的用户隐藏脚本？”答案是做不到。如果用户足够有毅力的话，他们总有办法查明你代码中的内容。

但是，如果你确实需要对一般水平的访问者隐藏代码，那么脚本 8-5 和脚本 8-6 可以防止访问者通过鼠标单击打开快捷菜单，从而防止他们查看页面的源代码。

脚本 8-5 你可能能查看这个页面的源代码，但是你需要做一些工作

```
<!DOCTYPE html>
<html>
<head>
     <title>onMousedown capture</title>
     <script src="script03.js"></script>
</head>
<body>
     <h1>Important source data that someone might want to look at.</h1>
</body>
</html>
```

脚本 8-6 这个脚本会阻止经验不太丰富的用户在页面上打开快捷菜单

```
if (typeof document.oncontextmenu == "object") {
     if (document.all) {
        document.onmousedown = captureMousedown;
     }
     else {
        document.oncontextmenu = captureMousedown;
     }
}
else {
     window.oncontextmenu = captureMousedown;
}

function captureMousedown(evt) {
     if (evt) {
        var mouseClick = evt.which;
     }
     else {
        var mouseClick = window.event.button;
     }

     if (mouseClick==1 ||  mouseClick==3) {
        alert("Menu Disabled");
        return false;
     }
}
```

1.
```
if (typeof document.oncontextmenu == "object") {
    if (document.all) {
       document.onmousedown = captureMousedown;
    }
```

第一块代码检查浏览器是否是 Firefox，这种浏览器使用 window.oncontextmenu（因此不知道 document.oncontextmenu）。如果它不是 Firefox，再检查 document.all，这是检查浏览器是否是 IE 的简便方法。如果它是 IE，就在触发 onmousedown 时运行 captureMousedown()。

```
2. else {
     document.oncontextmenu = captureMousedown;
   }
```

如果到达了这里，就说明访问者使用的是 Safari 或者 Chrome，这种浏览器需要在 document 对象上设置 oncontextmenu。

```
3. else {
     window.oncontextmenu = captureMousedown;
   }
```

最后，如果浏览器是 Firefox，就让 window 的 oncontextmenu 事件调用 captureMousedown()函数。

```
4. function captureMousedown(evt) {
```

这个函数要处理 onmousedown 和 oncontextmenu 事件。Chrome、Safari 和 Firefox 在触发事件时会自动地生成和传递 evt 参数，这个变量包含关于事件的信息。

```
5. if (evt) {
     var mouseClick = evt.which;
   }
   else {
     var mouseClick = window.event.button;
   }
```

如果 evt 变量存在，就可以通过检查 evt.which 来判断用户单击的是哪个按钮。如果用户使用 IE，用户操作的结果会在 window.event.button 中找到。无论是哪种情况，结果都存储在 mouseClick 变量中。

```
6. if (mouseClick==1 || mouseClick==3) {
     alert("Menu Disabled");
     return false;
   }
```

如果 mouseClick 是 1 或 3，就弹出一个警告框（见图 8-3），向用户指出这个功能已经禁用了，并且返回 false。返回 false 会阻止显示菜单窗口。

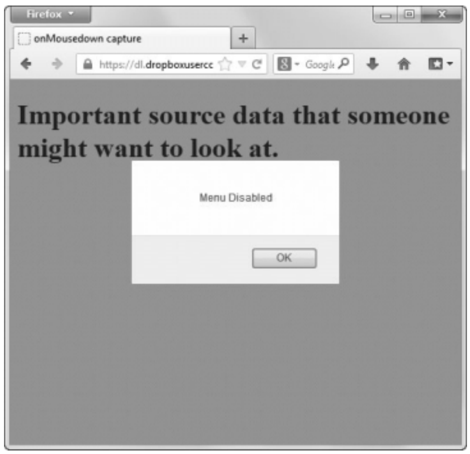


图 8-3　这个警告框会让一般用户放弃查看源代码的企图（也会给有经验的用户制造点儿障碍）

✔提示

- 我们为什么要检查 2 种不同的鼠标单击？检查一种难道不够吗？从理论上是够了，但是在实际上不够（见表 8-1）。问题是在 Mac 上，按下 Ctrl 键同时左键单击可能会触发快捷菜单，因此需要检查右键单击和左键单击两种情况。
- 但是，这种方法可能有负面效果：你可能成功地阻止了左键单击和右键单击，但是，这也意味着阻止用户单击页面上的任何链接。

表8-1　鼠标单击编码

编　码	事　件	浏览器
1	左键单击	IE 所有Mac浏览器
3	右键单击	所有浏览器

✓提示

- 对于有经验的访问者，回避这个脚本的控制是非常容易的：他们只需在浏览器中关闭 JavaScript 功能，单击功能就会恢复了。将 JavaScript 代码放在外部的.js 文件中似乎也能阻碍用户查看源代码，但是用户可以查看硬盘上的缓存文件夹。他们也可以查看页面的源代码，找到外部脚本文件的名称，然后在浏览器中输入外部文件的 URL，这样就可以正常地显示文件的内容。如果你确实很担心自己的源代码泄露出去，那么唯一真正可靠的保密方法是不把它放在 Web 上。
- IE 理解 `document.oncontextmenu`，所以你可能认为可以通过设置它来处理这些事件——但是不行。IE 是唯一需要设置 `document.onmousedown` 的浏览器。另外，如果同时设置 `window.oncontextmenu` 和 `document.onmousedown`，Firefox 会触发每个事件两次，每个操作一次。

8.2.2　onmouseup事件

与 `onmousedown` 事件相似，`onmouseup` 事件会在用户单击鼠标然后释放按钮时触发。

8.2.3　onmousemove事件

当页面的访问者移动鼠标时，就会触发 `onmousemove` 事件。在这个示例中，我们让用户觉得有一双眼睛一直盯着他们的鼠标移动（见图 8-4）。脚本 8-7、脚本 8-8 和脚本 8-9 演示如何使用 JavaScript 显示“一对眼睛”，它们会随着访问者移动鼠标而转动。

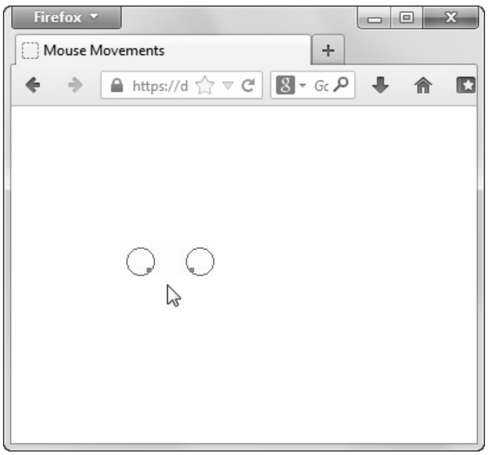


图 8-4　无论鼠标指针移动到哪里，这对眼睛都会一直盯着它

脚本 8-7　“紧跟着的眼睛”示例的 HTML

```
<!DOCTYPE html>
<html>
<head>
     <title>Mouse Movements</title>
     <link rel="stylesheet" href="script04.css">
     <script src="script04.js"></script>
</head>
<body>
     <img src="images/circle.gif" alt="left eye" id="lEye">
     <img src="images/circle.gif" alt="right eye" id="rEye">
     <img src="images/lilRed.gif" alt="left eyeball" id="lDot">
     <img src="images/lilRed.gif" alt="right eyeball" id="rDot">
</body>
</html>
```

脚本 8-8　“紧跟着的眼睛”示例的 CSS

```
body {
     background-color: #FFF;
}

#lEye, #rEye {
     position: absolute;
     top: 100px;
     width: 24px;
     height: 25px;
}

#lDot, #rDot {
     position: absolute;
     top: 113px;
     width: 4px;
     height: 4px;
}

#lEye {
     left: 100px;
}

#rEye {
     left: 150px;
}

#lDot {
     left: 118px;
}

#rDot {
     left: 153px;
}
```

脚本 8-9　这个脚本让眼睛一直盯着用户的鼠标指针

```
document.onmousemove = moveHandler;

function moveHandler(evt) {
     if (!evt) {
        evt = window.event;
     }
```

```
        animateEyes(evt.clientX, evt.clientY);
    }

    function animateEyes(xPos,yPos) {
        var rightEye = document.getElementById("rEye");
        var leftEye = document.getElementById("lEye");
        var rightEyeball = document.getElementById("rDot").style;
        var leftEyeball = document.getElementById("lDot").style;

        leftEyeball.left = newEyeballPos(xPos, leftEye.offsetLeft);
        leftEyeball.top = newEyeballPos(yPos, leftEye.offsetTop);
        rightEyeball.left = newEyeballPos(xPos, rightEye.offsetLeft);
        rightEyeball.top = newEyeballPos(yPos, rightEye.offsetTop);

        function newEyeballPos(currPos,eyePos) {
           return Math.min(Math.max(currPos,eyePos+3), eyePos+17) + "px";
        }
    }
```

1. `document.onmousemove = moveHandler;`

对于所有浏览器，如果触发了 mousemove 事件，就调用 moveHandler()函数。

2.
```
function moveHandler(evt) {
    if (!evt) {
      evt = window.event;
    }
    animateEyes(evt.clientX,evt.clientY);
  }
```

每当发生 mousemove 事件时，就会触发 moveHandler()函数。如果访问者使用 IE，就需要对 evt 进行初始化，然后对于所有浏览器，调用 animateEyes()函数并且将鼠标指针的 *X* 和 *Y* 坐标传递给它。

3. `function animateEyes(xPos,yPos) {`

这个函数根据传递给它的 *X* 和 *Y* 坐标转动眼睛。

4.
```
var rightEye = document.getElementById("rEye");
  var leftEye = document.getElementById("lEye");
  var rightEyeball = document.getElementById("rDot").style;
  var leftEyeball = document.getElementById("lDot").style;
```

这些代码将变量设置为左右眼眶图像和瞳孔图像的 id。

5.
```
leftEyeball.left = newEyeballPos(xPos, leftEye.offsetLeft);
  leftEyeball.top = newEyeballPos(yPos, leftEye.offsetTop);
  rightEyeball.left = newEyeballPos(xPos, rightEye.offsetLeft);
  rightEyeball.top = newEyeballPos(yPos, rightEye.offsetTop);
```

这些代码根据鼠标指针的位置绘制眼球，其中使用了下一步中定义的 newEyeballPos()函数的结果。

6.
```
function newEyeballPos(currPos,eyePos) {
    return Math.min(Math.max(currPos, eyePos+3),eyePos+17) + "px";
  }
```

我们不希望眼球跑到眼眶外边。所以对于每只眼球，我们要确保它尽可能接近鼠标指针，同时仍

然在眼眶范围内。

✔提示

- Web 上有一种常见的 JavaScript 效果：页面上有许多点（或者设计者希望的其他东西）一直跟随着鼠标指针的移动。我们不想重复已经存在的效果，所以设计了转动的眼睛。但是，如果你希望在页面上显示尾随鼠标指针的点，那么你应该能够通过修改这个脚本来实现。

8.2.4 onmouseover事件

现在，你应该很熟悉这个事件了：我们曾经利用它实现图像翻转器。当鼠标移动进任何注册了 onmouseover 事件处理程序的区域时，就会触发这个事件。

8.2.5 onmouseout事件

毫无疑问，既然有 onmouseover，当然也有对应的 onmouseout。当鼠标离开一个注册了此事件的区域时，就会触发这个事件。

8.2.6 ondblclick事件

因特网的缺点之一是，计算机用户已经习惯的用户界面元素在 Web 上改变了。例如，计算机新用户最早学习的操作就是用鼠标进行双击，但是在 Web 上没有双击操作，至少不曾有过。但是，现在可以使用脚本 8-10、脚本 8-11 和脚本 8-12 检查鼠标双击。

脚本 8-10 这个 HTML 帮助处理双击

```
<!DOCTYPE html>
<html>
<head>
     <title>Image Popup</title>
     <link rel="stylesheet" href="script05.css">
     <script src="script05.js"></script>
</head>
<body>
     <h3>Double-click on an image to see the full-size version</h3>
     <img src="images/Img0_thumb.jpg" alt="Thumbnail 0" id="Img0">
     <img src="images/Img1_thumb.jpg" alt="Thumbnail 1" id="Img1">
     <img src="images/Img2_thumb.jpg" alt="Thumbnail 2" id="Img2">
</body>
</html>
```

脚本 8-11 这段 CSS 使你的图像看起来更好看

```
body {
     background-color: #FFF;
}

img {
     margin: 0 10px;
     border: 3px #00F solid;
     width: 160px;
     height: 120px;
}
```

脚本 8-12　用这个脚本捕获和处理双击

```
window.onload = initImages;

function initImages() {
     for (var i=0; i<document.images.length;i++) {
        document.images[i].ondblclick = newWindow;
     }
}

function newWindow() {
     var imgName = "images/" + this.id + ".jpg"
     var imgWindow = window.open(imgName,"imgWin", "width=320,height=240,scrollbars=no")
}
```

- `document.images[i].ondblclick = newWindow;`

当用户双击缩略图之一时，就会触发 `newWindow()`事件处理函数。此时会弹出一个新窗口（见图 8-5），其中显示图像的大版本。



图 8-5　双击缩略图，就会打开该图像的大版本

8.2.7　onclick事件

`onclick` 处理程序的工作方式与 `ondblclick` 处理程序相似，差异仅仅是由单击触发它，而不是双击触发。`onmouseup` 处理程序也相似，差异是 `onclick` 要求用户按下鼠标按钮并放开才能触发，而 `onmouseup` 只需要后者。

8.3　表单事件处理

表单事件处理主要用来验证表单。通过使用下面列出的事件，可以处理用户在表单上所做的任何操作。

8.3.1　onsubmit事件

当用户单击 Submit 按钮来提交表单时，就会触发 `onsubmit` 处理程序（参见第 6 章）。另外，根据浏览器的不同，当用户退出表单上的最后一个文本输入字段时，也可能会触发它。如果脚本包含

onsubmit 处理程序，而且这个处理程序的结果是 false，那么表单就不会发送回服务器。

8.3.2 onreset事件

当用户单击表单上的 Reset 按钮（如果有这个按钮的话）时，就会触发 onreset 处理程序。如果表单具有在加载页面时设置的默认值，这会非常方便——如果用户单击 Reset 按钮，就需要用脚本动态地重新设置默认值。

8.3.3 onchange事件

如脚本 6-3 所示，当用户修改表单字段时，就会触发 onchange 事件处理程序。这可以用来立即验证输入的信息，或者在用户单击 Submit 按钮之前对用户的选择作出响应。

8.3.4 onselect事件

如果用户选择了一个 input 或 textarea 表单区域中的文本，就会触发 onselect 处理程序。

8.3.5 onclick事件

8.2 节提到了 onclick 处理程序，这里再次提到它是因为在处理表单时经常会用到它。如脚本 6-15 所示，当用户单击复选框或单选按钮时，就会触发这个事件。脚本 2-10 也使用了 onclick 处理程序。在那个示例中，它让一个链接对支持 JavaScript 的浏览器执行一种操作，而对不支持 JavaScript 的浏览器执行完全不同的另一种操作。

8.3.6 onblur事件

onblur 事件可以用于浏览器窗口（如前面所示），也经常用在表单上。脚本 8-13、脚本 8-14 和脚本 8-15 演示如何使用 onblur 处理程序迫使用户在一个字段中输入数据。

脚本 8-13 这个 HTML 创建一个简单的表单

```
<!DOCTYPE html>
<html>
<head>
     <title>Requiring an entry</title>
     <link rel="stylesheet" href="script06.css">
     <script src="script06.js"></script>
</head>
<body>
     <form action="#">
     <h3>
        Email address: <input type="text" class="reqd"><br><br>
        Name (optional): <input type="text">
     </h3>
     </form>
</body>
</html>
```

脚本 8-14　与 JavaScript 配合的 CSS

```
body {
     background-color: #FFF;
}

.highlight {
     background-color: #FF9;
}
```

脚本 8-15　当用户离开一个表单字段时，可以在表单中使用 onblur 处理程序触发操作

```
window.onload = initForm;

function initForm() {
     var allTags = document.forms[0].getElementsByTagName("*");

     for (var i=0; i<allTags.length; i++) {
        if (allTags[i].className.indexOf("reqd") > -1) {
           allTags[i].onblur = fieldCheck;
        }
     }
}

function fieldCheck() {
     if (this.value == "") {
        this.className += " highlight";
        this.focus();
     }
     else {
        this.className = "reqd";
     }
}
```

1. `if (allTags[i].className.indexOf("reqd") > -1) {`

我们使用 class 属性（reqd）在运行时决定什么时候应该使用 onblur 事件处理程序。只需在输入标签中添加 class="reqd"就可以触发这个事件，而不需要在各个字段上分别设置 onblur 处理程序。

2. `allTags[i].onblur = fieldCheck;`

当用户离开必须填写的字段时，字段上的这个事件处理程序会调用 fieldCheck()函数。

3.
```
function fieldCheck() {
    if (this.value == "") {
       this.className += " highlight";
       this.focus();
    }
    else {
       this.className = "reqd";
    }
  }
```

fieldCheck()函数检查在当前字段中是否输入了某些信息。如果这个字段还没有值，那么通过将"highlight"添加到它的 class 属性上，将字段的背景颜色改为浅黄色（见图 8-6），并且用 focus()将焦点重新放回这个表单字段中。纠正了错误之后，简单地将 class 属性重新设置为其初始值，背景颜

色就恢复为白色。

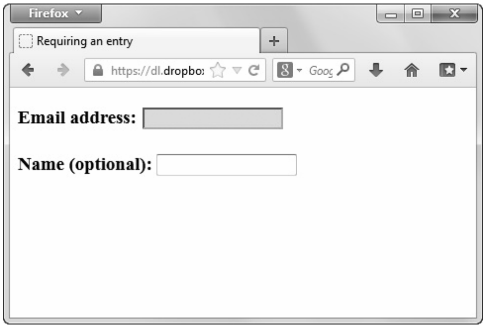


图 8-6 如果用户离开 Email address 字段，但是没有输入任何内容，那么字段变成黄色的并且保持活跃，直到输入了数据为止

✔提示

- ❑ 当用户在改变字段之后离开它时，会触发 onblur 和 onchange 两种事件。如果用户在没有改变字段内容的情况下离开，就只触发 onblur 处理程序。
- ❑ 当前的 Firefox 版本在 focus()方面有一个问题：尽管 focus()要求浏览器将焦点留在一个字段中，但是 Firefox 不这么做。但是，背景颜色的改变会提醒用户出了错误，所以他们仍然知道这个字段有问题。

8

8.3.7 onfocus事件

有时候，页面上的某个表单字段包含只读数据，这一数据要在表单上显示，但是不希望用户修改它。可以使用 HTML 属性 readonly 避免用户修改字段，但是并非所有浏览器都支持这个属性。脚本 8-16 和脚本 8-17 演示如何使用 onfocus 事件迫使用户离开这个字段，从而避免用户修改他们不应该修改的字段。

脚本 8-16 这个 HTML 创建一个表单，其中的电子邮件字段不允许修改

```
<!DOCTYPE html>
<html>
<head>
     <title>Forbidding an entry</title>
     <script src="script07.js"></script>
</head>
<body>
     <form action="#">
     <h3>
        Your message: <textarea rows="5" cols="30">Enter your message here </textarea>
        <br><br>
        Will be sent to: <input type="text" value="js9@javascriptworld.com" readonly size="25" />
     </h3>
     </form>
</body>
</html>
```

脚本 8-17　在表单上使用 onfocus 处理程序防止任意修改数据

```
window.onload = initForm;

function initForm() {
     var allTags = document.forms[0].getElementsByTagName("*");

     for (var i=0; i<allTags.length; i++) {
        if (allTags[i].readOnly) {
           allTags[i].onfocus = function() {
              this.blur();
           }
        }
     }
}
```

- ```
  allTags[i].onfocus = function() {
       this.blur();
    }
  ```

当用户试图进入这个字段时，焦点（本例中是活动的字段）会立刻自动地转移（见图 8-7）。这是因为 onfocus 事件处理程序调用一个匿名函数（没有名字的函数），这个函数只做一件事：在当前字段上调用 blur()，从而使焦点转移。

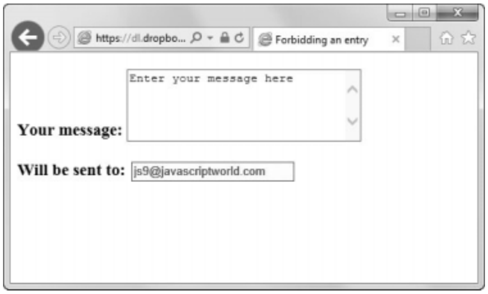


图 8-7　用户无法在下面的字段中输入任何信息

8.4　键事件处理

除了鼠标之外，另一种主要的输入设备是键盘。除非以后出现直接用思想控制的计算机设备，否则计算机还是离不开键盘。与鼠标一样，JavaScript 也为处理键盘提供了一些事件。

8.4.1　onkeydown事件

如果用户能够用键盘和鼠标两种方式控制网页，那么会很方便。通过使用键事件处理程序，可以在用户按下适当的键时执行相应的操作。在脚本 8-18、脚本 8-19 和脚本 8-20 中，通过按键盘上的左右箭头键，可以查看标准的幻灯片（与脚本 4-19 中使用的幻灯片相同），见图 8-8。
  ```

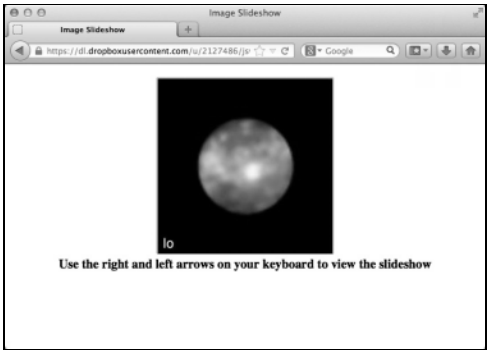


图 8-8　这个幻灯片由键盘控制，而不是由鼠标单击导航按钮来控制

脚本 8-18　幻灯片的 HTML

```
<!DOCTYPE html>
<html>
<head>
     <title>Image Slideshow</title>
     <link rel="stylesheet" href="script08.css">
     <script src="script08.js"></script>
</head>
<body>
     <h3 class="centered">
        <img src="images/callisto.jpg" id="myPicture" alt="Slideshow"><br>
        Use the right and left arrows on your keyboard to view the slideshow
     </h3>
</body>
</html>
```

脚本 8-19　CSS 再次使图像看起来更好看

```
body {
     background-color: #FFF;
}

.centered {
     text-align: center;
}

img#myPicture {
     width: 262px;
     height: 262px;
}
```

脚本 8-20　在这个脚本中使用 onkeydown 处理程序执行幻灯片的切换

```
document.onkeydown = keyHit;
var thisPic = 0;

function keyHit(evt) {
     var myPix = new Array("images/callisto.jpg", "images/europa.jpg","images/io.jpg", "images/
     →ganymede.jpg");
     var imgCt = myPix.length-1;
     var ltArrow = 37;
```

8

```
        var rtArrow = 39;

        if (evt) {
            var thisKey = evt.which;
        }
        else {
            var thisKey = window.event.keyCode;
        }

        if (thisKey == ltArrow) {
            chgSlide(-1);
        }
        else if (thisKey == rtArrow) {
            chgSlide(1);
        }

        function chgSlide(direction) {
            thisPic = thisPic + direction;
            if (thisPic > imgCt) {
                thisPic = 0;
            }
            if (thisPic < 0) {
                thisPic = imgCt;
            }
            document.getElementById("myPicture").src = myPix[thisPic];
        }
    }
```

1. `document.onkeydown = keyHit;`

这里将 keyHit()函数注册为 onkeydown 事件处理程序。

2. `var thisPic = 0;`

在全局范围对变量 thisPic 进行初始化和设置，因此它存储全局的值，在每次调用 keyHit()时都可以使用这个变量。

3. `function keyHit(evt) {`

keyHit()函数处理击键事件。

4.
```
var ltArrow = 37;
var rtArrow = 39;
```

左箭头键产生数字 37，右箭头键产生 39。我们将这些值存储在变量中，用来判断用户按下的是哪个键。

5.
```
if (evt) {
    var thisKey = evt.which;
}
else {
    var thisKey = window.event.keyCode;
}
```

了解用户按下哪个键的方法取决于他们使用的浏览器。如果是 Firefox、Chrome 或 Safari，就查看 evt.which，这个属性包含键的编码。如果是 IE，那么编码包含在 window.event.keyCode 中。无论是哪种情况，都将结果保存在 thisKey 中。

```
6. if (thisKey == ltArrow) {
     chgSlide(-1);
   }
   else if (thisKey == rtArrow) {
     chgSlide(1);
   }
```

如果用户按下左箭头键，就将幻灯片退后一帧；如果按下右箭头键，就将幻灯片前进一帧。如果他们按下的是其他键，就不执行任何操作。

✔提示

- 如果不确定某个键的键值，可以在步骤 5 和步骤 6 的代码行之间加上 `alert(thiskey);`，然后按下你要检查的键。这个警告框会显示键值。

8.4.2 onkeyup事件

onkeyup 事件处理程序与 onkeydown 处理程序相同，唯一的差异是，它在用户已按下并释放键的过程中触发。

8

8.4.3 onkeypress事件

当用户按下并释放键时触发 onkeypress 事件。

8.5 高级事件处理

除了本书前面提到的事件处理，还有另外一种事件处理模型，通常称为 DOM Level 2 事件处理程序。这种更为灵活的新方法有不少优点。

- 有一种通用的方法设置几类事件处理程序。
- 可为单个事件注册多个事件处理程序，也就是说添加其他事件处理程序不会覆写已有的事件处理程序。
- 能够控制事件冒泡或者捕获（请查看补充内容“冒泡和捕获”）。
- 事件可触发其他事件。
- 事件处理程序很容易被移除。

✔提示

- 第 10 章会详细讲解文档对象模型（DOM）。

8.5.1 addEventListener方法

这里的任务跟前一个任务一样，只是这个事件处理程序是通过 addEventListener()设置的，在脚本 8-21 中可以看到，结果如图 8-9 所示。

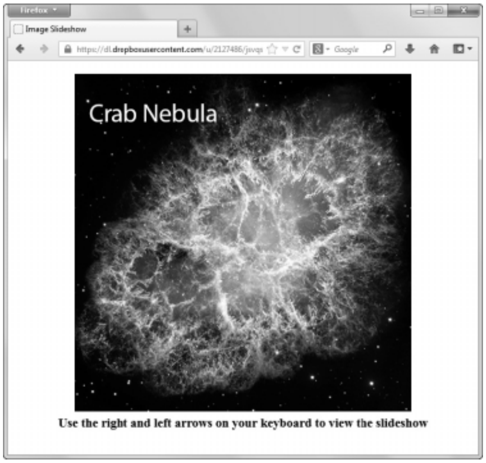


图 8-9　这个幻灯片和前面的幻灯片功能一样，只是我们用了另外一种方式来注册事件处理程序

脚本 8-21　JavaScript 代码非常相似，只是这里使用的是 addEventListener()函数

```
document.addEventListener("keydown",keyHit,false);
var thisPic = 0;

function keyHit(evt) {
     var myPix = new Array("images/catseyenebula.jpg", "images/crabnebula.jpg","images/eskimonebula.jpg",
     ➝"images/ringnebula.jpg");
     var imgCt = myPix.length-1;
     var ltArrow = 37;
     var rtArrow = 39;

     if (evt) {
        var thisKey = evt.which;
     }
     else {
        var thisKey = window.event.keyCode;
     }

     if (thisKey == ltArrow) {
        chgSlide(-1);
     }
     else if (thisKey == rtArrow) {
        chgSlide(1);
     }

     function chgSlide(direction) {
        thisPic = thisPic + direction;
        if (thisPic > imgCt) {
           thisPic = 0;
        }
        if (thisPic < 0) {
           thisPic = imgCt;
        }
        document.getElementById("myPicture").src = myPix[thisPic];
     }
}
```

- ❑ document.addEventListener("keydown",keyHit,false);

此处，我们注册 keyHit()函数来处理 onkeydown 事件。addEventListener()函数有 3 个参数：事件本身（目标）、触发事件时调用的函数（监听器），以及用来指定事件被捕获（true）还是冒泡（false）的布尔值。

✔提示

- ❑ 传递给 addEventListener()的目标事件跟本章其他地方使用的事件一样，省略了开头的 on。

8.5.2 removeEventListener方法

该方法允许从它的目标事件移除事件监听器。

8.5.3 dispatchEvent方法

该方法允许从代码中的其他位置触发事件处理程序。它接收一个参数：Event 对象。例如，如果要创建、初始化并分派一个事件来点击链接，代码可以这样：

```
var evt = document.createEvent("Event");
evt.initEvent("click", true, false);
document.getElementById("theLink").dispatchEvent(evt);
```


8.5.4 initEvent方法

该方法初始化已创建的事件（通常是通过调用 document.createEvent("Event");来实现的）。它接收三个参数：事件类型、表示事件是否冒泡的布尔值，以及表示事件能否被取消的布尔值。

✔提示

- ❑ 表示事件冒泡的布尔值是 false，initEvent()是个例外，它用 true 表示冒泡。

8.5.5 stopPropagation方法

该方法阻止触发事件流中的其他事件，它没有参数。

8.5.6 preventDefault方法

该方法取消正在进行的事件（如果该事件是可取消的），它没有参数。

冒泡和捕获

我们可以将网页的结构想象为一棵树。比如说网页的主体可以包含表格，表格本身又包含多行，而表格行内包含了若干单元格，见图 8-10。

比方说，你编写了一个脚本，一旦鼠标移过页面上的元素，就会弹出提示框（alert）。如果移过单元格，你希望弹出"移过单元格"的提示，你肯定能收到。不仅如此，随后你还会收到"移过行"、"移过表格"的提示，最后是"移过主体"。这就是**事件冒泡**，也就是说事件从树的下面向上面冒泡。这是网页工作的一般方式。

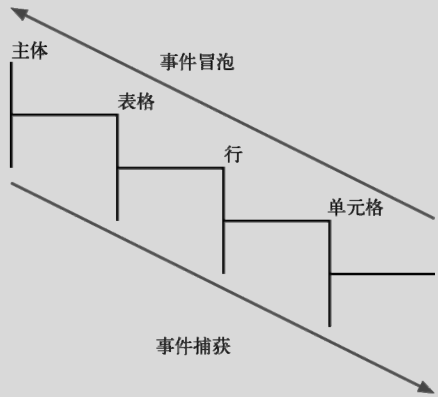


图 8-10　可以从下向上触发事件（冒泡），也可以从上向下触发事件（捕获）

但是，如果你想让事件以相反的方向发生呢，也就是从上向下？这种情况就称为**事件捕获**，脚本弹出提示框的顺序是主体、表格、行，最后是单元格。

这是使用高级事件处理程序的一大优势：调用 addEventListener()的时候，开发人员可以控制事件发生的方向。

第 9 章

JavaScript 和 cookie

在 Web 术语中，cookie 是一小段信息，当用户第一次访问 Web 服务器时，服务器将这些信息发送给浏览器。这个用户以后每次访问这个 Web 站点时，Web 服务器可以通过 cookie 识别这个用户。浏览器将 cookie（其中包含关于访问者的信息）作为纯文本文件保存在计算机硬盘上。

作为 JavaScript 开发人员，你可以用 cookie 做许多有意义的事情。如果你的站点要求注册，那么可以用 cookie 将访问者的用户名和密码保存在他们的硬盘上，这样他们就不需要在每次访问时都输入用户名和密码。可以跟踪用户已经访问过站点的哪些部分，以及统计用户的访问次数。

关于 cookie 有许多常见的误解，所以一定要知道 cookie 不能实现哪些操作：无法获得关于用户的任何真实信息，比如他们的电子邮件地址；无法使用 cookie 查看用户硬盘上的内容；cookie 也无法传输计算机病毒。cookie 只是用户硬盘上一个简单的文本文件，JavaScript 开发人员可以在其中存储一些信息，仅此而已。

cookie 总是包含发送它的服务器的地址。cookie 技术背后的本质是"识别"。可以把它看作 Web 上的 Caller ID，只是在形式方面有各种变化——每个使用 cookie 的 Web 站点向用户的浏览器授予某种形式的个性化 ID，这样在用户下一次访问这个站点时就能够识别出他。当用户再次访问以前向他传递过 cookie 的 Web 服务器时，服务器可以查询浏览器，了解这个用户是否拥有它的 cookie。如果是，服务器就可以获取原来传递的 cookie 中存储的信息。请记住，cookie 只识别使用的计算机，而不识别使用这台计算机的人。

9.1　建立第一个 cookie

cookie 是一个具有特定格式的文本字符串：

```
cookieName=cookieValue;expires=expirationDateGMT;path=URLpath;domain=siteDomain
```

这个字符串的第一部分给 cookie 命名并给它赋值。这是 cookie 中唯一必须有的部分，字符串的其余部分都是可选的。接下来是 cookie 的过期日期（expiration date），当到了这个日期，浏览器会自动删除这个 cookie。过期日期后面是一个 URL 路径，这允许在 cookie 中存储一个 URL。最后，可以在 cookie 中存储一个域值。

脚本 9-1 中的 HTML 文件调用脚本 9-2 中的 JavaScript，它会根据用户在表单中输入的值设置 cookie。在使用这个页面时，表面上看不出什么效果（见图 9-1），但是在幕后确实创建了 cookie。本章后面的示例将在这个示例的基础上进行扩展。



图 9-1 虽然在表面上看不出来，但是表单文本字段的内容已经被写入 cookie 中

脚本 9-1 第一个 cookie 页面的 HTML

```
<!DOCTYPE html>
<html>
<head>
     <title>Set a cookie based on a form</title>
     <script src="script01.js"></script>
</head>
<body>
     <form id="cookieForm" action="#">
        <h1>Enter your name: <input type="text" id="nameField"></h1>
     </form>
</body>
</html>
```

脚本 9-2 使用这个脚本设置浏览器 cookie

```
window.addEventListener("load",nameFieldInit,false);

function nameFieldInit() {
     var userName = "";
     if (document.cookie != "") {
        userName = document.cookie.split("=")[1];
     }

     document.getElementById("nameField").value = userName;
     document.getElementById("nameField").onblur = setCookie;
     document.getElementById("cookieForm").onsubmit = setCookie;
}

function setCookie() {
     var expireDate = new Date();
     expireDate.setMonth(expireDate.getMonth()+6);

     var userName = document.getElementById("nameField").value;
     document.cookie = "userName=" + userName + ";expires=" + expireDate.toGMTString();

     document.getElementById("nameField").blur();
     return false;
}
```

⇒ 设置 cookie

1. `function nameFieldInit() {`

首先创建函数 nameFieldInit()来定义 cookie 的值。当窗口完成加载时，会调用这个函数。

2. var userName = "";

接下来，用一个空值对变量 userName 进行初始化。

3. if (document.cookie != "") {
    userName = document.cookie.split("=")[1];

这个条件测试检查 document.cookie 对象是否不包含空值。split("=")方法将 cookie 分隔成一个数组，数组中第一个元素（cookieField[0]）是 cookie 的名称，第二个元素（cookieField [1]）是 cookie 的值。注意，该数组可以是存储 cookie 字段的任何变量。这里将 document.cookie.split("=")[1]返回的值（也就是 cookie 的值）赋值给 userName。

4. document.getElementById("nameField").value = userName;

如果 cookie 文件中存储了一个用户名，那么在页面加载时设置 getElementById ("nameField").value 会把这个用户名放进文本字段中。

5. document.getElementById("nameField").onblur = setCookie;
  document.getElementById("cookieForm").onsubmit = setCookie;

在第一行中，当用户离开这个文本字段时，onblur 事件处理程序（见第 1 章和第 8 章）会调用 setCookie()函数。在第二行，表单的 onsubmit 事件处理程序做同样的事情。如果用户在输入姓名后按 Enter 键，IE 不会触发 onblur 处理程序（由于某些原因）。添加 onsubmit 处理程序是为了适应所有浏览器。

6. function setCookie() {

现在创建一个新函数 setCookie()。

7. var expireDate = new Date();

获得当前日期并且将它赋值给新变量 expireDate。

8. expireDate.setMonth(expireDate.getMonth()+6);

这一行取出 expireDate 的月份部分，在月份上加 6，然后将 expireDate 的月份部分设置为新的值。换句话说，它将 cookie 的过期日期设置为创建 cookie 之后 6 个月。

9. var userName = document.getElementById("nameField").value;

这一行创建一个新变量 userName，并且将用户在文本字段中输入的值赋给它。userName 变量必须创建两次（每次在不同的函数中），因为它不是全局变量，也就是说，可以在每个函数中使用它，但是并不指望跨函数保持它的值——在每个函数中它的值都是新的。

10. document.cookie = "userName=" + userName + ";expires=" + expireDate.toGMTString();

这里是写 cookie 的地方。请记住，cookie 仅仅是一个文本字符串，所以可以使用任何文本字符串操作来建立 cookie，比如使用加号来连接字符串。我们设置 document.cookie 以包含用户名和 cookie 过期日期。toGMTString()方法将 Date 对象 expireDate 转换为文本字符串，以便将它写入 cookie 中。

11. document.getElementById("nameField").blur();
   return false;

那么，如何设置表单，从而允许以两种方式调用 setCookie()？下面是处理这个问题的方法。

- 如果浏览器是 IE，那么第一行使焦点离开 name 字段，从而表明发生了某些情况。第二行（返回 false 值）防止实际提交表单。

9

- 如果浏览器不是 IE，那么第一行没有任何作用（我们已经离开了 name 字段，所以再次离开它不会有影响）。第二行防止触发表单提交。

多个 cookie

可以在一个页面上设置多个 cookie，格式如下：

```
"cookieName1=cookieValue1;expires=expirationDateGMT1;path=sitePath1;domain=siteDomain1";

"cookieName2=cookieValue2;expires=expirationDateGMT2;path=sitePath2;domain=siteDomain2"
```

同样，只有名称和值对是必须有的字段。

split("; ")命令将多个 cookie 记录分隔为数组，各个 cookie 从零开始编号。注意，这个命令中分号后面有一个空格。cookieArray[0]是多个 cookie 记录中的第一个 cookie，cookieArray[1]是下一个 cookie，以此类推。更多细节请参见本章后面的 9.6 节。

✔提示

- 这个脚本假设第一个 cookie 包含用户名。后面的脚本将演示如何处理多个 cookie，并且根据名称（而不是编号）来获得 cookie。
- 如果你正处在下半年，无需担心遭遇日期重置问题。为月份加 6 总是会生成正确的结果，也就是说，下半年的情况下，年份加 1 的同时，月份会减 6。
- 本章中的脚本具有特定的次序，如果你按照它们在书中出现的次序运行它们，那么它们会正常运行。但是，如果打乱次序，就可能遇到某些怪异的结果（比如，浏览器认为你的姓名是一个数字）。如果希望不按次序运行它们，就应该在脚本的各次执行之间运行脚本 9-7。

9.2　读取 cookie

设置了 cookie 之后，需要获得它以便做某些有意义的事情。前一个示例在 cookie 中设置了文本字符串 Tom。脚本 9-3 和脚本 9-4 演示如何从 cookie 中获得这个值并且将它显示在屏幕上（当然，一般情况下不需要显示 cookie，这个脚本显示 cookie 只是为了演示）。

脚本 9-3　JavaScript 将使用这个 HTML 页面中的 id 来插入 cookie 结果

```
<!DOCTYPE html>
<html>
<head>
     <title>I know your name!</title>
     <script src="script02.js"></script>
</head>
<body>
     <h1 id="nameField"> </h1>
</body>
</html>
```

脚本 9-4　这个简短的脚本读取前面设置的 cookie 并且将它发送到文档窗口

```
window.addEventListener("load",nameFieldInit,false);
```

```
function nameFieldInit() {
     if (document.cookie != "") {
        document.getElementById("nameField").innerHTML = "Hello, " + document.cookie.split("=")[1];
     }
}
```

⇛ 读取 cookie

1. `if (document.cookie != "") {`

确保 `document.cookie` 对象中的值非空。

2. `document.getElementById("nameField").innerHTML = "Hello, " + document.cookie.split("=")[1];`

如果 cookie 非空，就向屏幕上写一个文本字符串：`"Hello,"`（注意逗号后面有空格），再加上提取出的 cookie 值（见图 9-2）。



图 9-2　这个 cookie 提供了我的姓名

✔提示

- 你是否注意到我们不需要指定要读取 cookie 文件中的哪个 cookie？这是因为一个 cookie 只能由最初写它的服务器读取。浏览器内部的 cookie 机制不允许你读或写别人所写的 cookie。你只能访问自己的 cookie。

9.3　显示 cookie

在前一个示例中，我们读取了一个来自服务器的 cookie 的值。现在，我们看看如何编写一个脚本，让它读取来自你的服务器的所有 cookie，并且显示它们的名称和值。如果没有 cookie，脚本就显示 There are no cookies here（见图 9-3）。如果有 cookie，就将每个 cookie 的内容显示在单独的行上（见图 9-4）。脚本 9-5 演示具体做法。



图 9-3　如果来自网页所在的服务器没有 cookie，就会看到这个结果

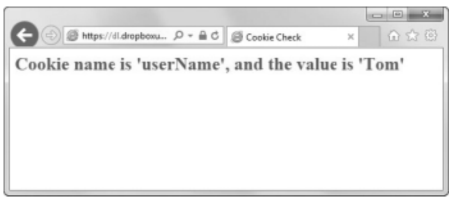


图 9-4　如果有一个或多个 cookie，那么脚本将它们写到文档窗口

脚本 9-5　这个脚本循环遍历并且显示某个特定服务器在你的机器上设置的所有 cookie

```
window.addEventListener("load",showCookies,false);

function showCookies() {
     var outMsg = "";

     if (document.cookie == "") {
        outMsg = "There are no cookies here";
     }
     else {
        var thisCookie = document.cookie.split("; ");

        for (var i=0; i<thisCookie.length;i++) {
           outMsg += "Cookie name is '" + thisCookie[i].split("=")[0];
           outMsg += "', and the value is '" + thisCookie[i].split("=") [1] + "'<br>";
        }
     }
     document.getElementById("cookieData").innerHTML = outMsg;
}
```

⇛ 显示所有的 cookie

1. `var outMsg = "";`

首先对变量 outMsg 进行初始化，这个变量将包含我们要显示的消息。

2. `if (document.cookie == "") {`

    `outMsg = "There are no cookies here";`

这个条件测试的意思是：如果 document.cookie 是空的，那么将 outMsg 设置为 There are no cookies here。

3. `var thisCookie = document.cookie.split("; ");`

如果前面的测试失败（即至少有一个 cookie 存在），那么使用 document.cookie.split("; ")获得所有 cookie 的值，并且将这些值存储进数组 thisCookie。请记住，split("; ")命令创建一个包含所有 cookie 的数组。然后，脚本就能够引用这个数组中的每个值。

4. `for (var i=0; i<thisCookie.length;i++) {`

这一行开始一个循环，首先将计数器变量 i 的值设置为零。然后，设置循环条件为“i 小于 thisCookie 数组中的 cookie 数量”。最后将 i 的值递增 1。

5. `outMsg += "Cookie name is '" + thisCookie[i].split("=")[0]);`

   `outMsg += "', and the value is '" + thisCookie[i].split("=")[1] + "'<br>");`

在脚本遍历数组时，对于每个 cookie，它将文本字符串"Cookie name is '"写入 outMsg，后跟 cookie 的名称。然后加上文本字符串"', and the value is '"和 cookie 的值。最后，在每行的末尾加上一个 HTML 分隔符。

6. `document.getElementById.("cookieData").innerHTML = outMsg;`

在遍历所有 cookie 之后，变量 outMsg 已经设置好了，所以通过 innerHTML 将它写到页面上。

9.4　使用 cookie 作为计数器

因为 cookie 是持久性的，也就是说，它们可以跨 Web 服务器和浏览器之间的多次会话持久地存在，所以可以使用 cookie 存储特定用户访问某个页面的次数。但是，这并不是在许多网页上看到的页

面计数器。因为 cookie 是与一个用户相关联的，所以只能记录这个用户的访问次数，不能使用 cookie 存储所有用户访问这个页面的次数。但是，了解如何创建这样的个人计数器是有意义的，还可以修改脚本 9-6 来完成其他任务（参见提示）。

脚本 9-6 这个脚本在 cookie 中记录访问次数

```
window.addEventListener("load",initPage,false);

function initPage() {
     var expireDate = new Date();
     expireDate.setMonth(expireDate.getMonth()+6);

     var hitCt = parseInt(cookieVal("pageHit"));
     hitCt++;

     document.cookie = "pageHit=" + hitCt + ";expires=" + expireDate.toGMTString();
     document.getElementById("pageHits").innerHTML = "You have visited this page " + hitCt + " times.";
}

function cookieVal(cookieName) {
     var thisCookie = document.cookie.split("; ");

     for (var i=0; i<thisCookie.length; i++) {
        if (cookieName == thisCookie[i].split("=")[0]) {
           return thisCookie[i].split("=")[1];
        }
     }
     return 0;
}
```

9

⇛ 使用 cookie 作为计数器

1. `var expireDate = new Date();`
   `expireDate.setMonth(expireDate.getMonth()+6);`
   这两行与 9.1 节中的步骤 7 和步骤 8 相同。请参见那里的解释。

2. `var hitCt = parseInt(cookieVal("pageHit"));`
   字符串 pageHit 是这个 cookie 的名称。在后面的步骤中，你会看到 cookieVal()函数的内容。这一行从 cookieVal()获得 cookie 的名字，并且用 parseInt()方法将它转换为数字，然后将结果存储进变量 hitCt。parseInt()方法将一个字符串（cookie 中的值）转换为数字（用作计数器的值）。

3. `hitCt++;`
   现在将 hitCt 的值加 1，从而递增计数器。

4. `document.cookie = "pageHit=" + hitCt + ";expires=" + expireDate.toGMTString();`
   这行代码将更新后的信息写回 cookie。写入的内容是一个文本字符串，其中包括字符串"pageHit="、递增后的 hitCt 值、";expires="和过期日期（这个日期在步骤 1 中设置为当前日期后的 6 个月）。

5. `document.getElementById("pageHits").innerHTML = "You have visited this page " + hitCt + "times.";`
   这一行在文档窗口中显示用户消息（见图 9-5）。在"page"后面和"times"前面有额外的空格，这会使消息在屏幕上看起来意义更明确。



图 9-5　页面指出我们已经访问了它 4 次，真是难以置信

6. `function cookieVal(cookieName) {`

这一行创建新函数 `cookieVal()`。需要向它传递一些数据，在函数中可以用变量 `cookieName` 引用这些数据。

7. `var thisCookie = document.cookie.split("; ");`

将变量 `thisCookie` 设置为 `split("; ")`方法生成的数组。

8. `for (var i=0; i<thisCookie.length;i++) {`

现在开始一个循环，这与 9.3 节中的步骤 4 相同。

9. `if (cookieName == thisCookie[i].split("=")[0]) {`

这个条件语句检查 `cookieName` 是否与 cookie 数组中的第 `i` 个元素的名称相同。

10. `return thisCookie[i].split("=")[1];`

如果步骤 9 中的测试成功了，那么返回 cookie 的值。

11. `return 0;`

如果检查了数组中的所有元素，但是没有找到匹配，就返回 0 值。

✔提示

- ❑ 当对调用这个脚本的 HTML 页面进行加载时，在浏览器中单击 Reload 按钮就会看到计数器递增。
- ❑ 如前所述，可以修改脚本 9-6 来完成其他任务。一种可能性是使用 cookie 记录特定用户上一次访问你站点的时间，并且根据这个时间显示不同的页面。例如，有些在线杂志有一个封面页面，其中显示插图和当天的新闻标题。如果用户在 24 小时内多次访问这个站点，那么他们只会在第一次看到封面页面，后续的访问会将用户直接转到站点的目录页面。
- ❑ 如果你需要真正的页面访问计数器，即显示所有用户加载某个页面的总次数的计数器，那么需要在 Web 服务器上安装一个计数器程序。可以向你的 Web 主机托管公司询问他们是否提供计数器功能，也可以在搜索引擎中搜索“Web page hit counter”。

9.5　删除 cookie

有时候，你希望删除 cookie 记录中的一个或多个 cookie。这非常容易，一种效果很好的技术是，将 cookie 的过期日期设置为过去的某个日期，这会让浏览器自动地删除它。脚本 9-7 演示如何迫使 cookie 马上过期。

脚本 9-7　这个脚本删除 cookie

```
window.addEventListener("load",cookieDelete,false);

function cookieDelete() {
     var cookieCt = 0;

     if (document.cookie != "" && confirm ("Do you want to delete the cookies?")) {
        var thisCookie = document.cookie.split("; ");
        cookieCt = thisCookie.length;

        var expireDate = new Date();
        expireDate.setDate(expireDate.getDate()-1);

        for (var i=0; i<cookieCt; i++) {
           var cookieName = thisCookie[i].split("=")[0];
           document.cookie = cookieName + "=;expires=" + expireDate.toGMTString();
        }
     }
     document.getElementById("cookieData").innerHTML = "Number of cookies deleted: " + cookieCt;
}
```

⇛ 删除 cookie

1. `var cookieCt = 0;`

这个脚本要记录我们已经删除了多少个 cookie，所以首先创建 `cookieCt` 变量并且将它设置为零。

2. `if (document.cookie != "" && confirm("Do you want to delete the cookies?")) {`

这个测试首先确保 cookie 不包含空值，也就是说，存在一些 cookie。如果测试表明 cookie 是空的，那么脚本就不进行任何操作。测试的第二部分让浏览器弹出一个确认对话框，其中显示函数调用中的文本（见图 9-6）。如果 `confirm()`返回 `true`，我们就知道用户希望删除他们的 cookie，如果返回 `false`，就跳到步骤 9。

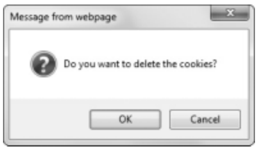


图 9-6　每当你要删除任何东西时，都要让用户确认，这是一种良好的界面设计习惯

3. `var thisCookie = document.cookie.split("; ");`

这一行用 `split("; ")`方法将 cookie 的内容分隔成数组，并且将这个数组赋值给变量 `thisCookie`。

4. `cookieCt = thisCookie.length;`

我们现在知道将要删除多少个 cookie，所以将这个值存储在 `cookieCt` 中。

5. `var expireDate = new Date();`
   `expireDate.setDate(expireDate.getDate()-1);`

这里创建一个新的日期对象 `expireDate`，然后将它设置为当前日期减 1——换句话说，就是昨天。

6. `for (var i=0; i<cookieCt; i++) {`

现在开始一个 `for` 循环，这样就可以删除所有 cookie，而不只是一个。首先将 `i` 的值设置为零，只要 `i` 小于 cookie 的数量，就将 `i` 递增 1。

7. var cookieName = thisCookie[i].split("=")[0];

使用 split("=")[0]获得数组中第 i 个 cookie 的名称，然后将它存储在变量 cookieName 中。

8. document.cookie = cookieName + "=;expires=" + expireDate.toGMTString();

这里将修改后的过期日期写回 cookie 中。

9. document.getElementById("cookieData").innerHTML = "Number of cookies deleted: "+ cookieCt;

脚本现在已经离开了 for 循环，这一行将已经删除的 cookie 数量写到 HTML 文档中（见图 9-7）。



图 9-7　用户也应该获得已经发生的事件的反馈

9.6　处理多个 cookie

常常希望同时处理多个 cookie，脚本 9-8 演示如何读取多个 cookie 并且显示其中的信息。这个示例重用了 9.4 节中的大量代码。

脚本 9-8　在一个脚本中使用数组处理多个 cookie

```
window.addEventListener("load",initPage,false);

function initPage() {
     var now = new Date();
     var expireDate = new Date();
     expireDate.setMonth(expireDate.getMonth()+6);

     var hitCt = parseInt(cookieVal("pageHit"));
     hitCt++;

     var lastVisit = cookieVal("pageVisit");
     if (lastVisit == 0) {
        lastVisit = "";
     }

     document.cookie = "pageHit=" + hitCt + ";expires=" + expireDate.toGMTString();
     document.cookie = "pageVisit=" + now + ";expires=" + expireDate.toGMTString();

     var outMsg = "You have visited this page " + hitCt + " times.";
     if (lastVisit != "") {
        outMsg += "<br>Your last visit was " + lastVisit;
     }
```

```
        document.getElementById("cookieData").innerHTML = outMsg;
    }

    function cookieVal(cookieName) {
        var thisCookie = document.cookie.split("; ");

        for (var i=0; i<thisCookie.length; i++) {
            if (cookieName == thisCookie[i].split("=")[0]) {
                return thisCookie[i].split("=")[1];
            }
        }
        return 0;
    }
```

⇒ 处理多个 cookie

1. `var lastVisit = cookieVal("pageVisit");`

首先将字符串 pageVisit 传递给 cookieVal()函数，从而寻找名为 pageVisit 的 cookie。它返回一个值，然后将这个值存储在 lastVisit 中。

2.
```
if (lastVisit == 0) {
    lastVisit = "";
}
```

如果 lastVisit 的值是零，就将空值放进 lastVisit。我们现在知道这个用户以前没有访问过这里。

3.
```
document.cookie = "pageHit=" + hitCt + ";expires=" + expireDate.toGMTString();
document.cookie = "pageVisit=" + now + ";expires=" + expireDate.toGMTString();
```

这两行将两个 cookie 写回硬盘，其中的信息包括更新后的单击次数和访问时间。

4.
```
var outMsg = "You have visited this page " + hitCt + " times.";
if (lastVisit != "") {
    outMsg += "<br>Your last visit was " + lastVisit;
}
```

outMsg 变量存储要向站点访问者显示的消息。其中的第一部分显示用户的访问次数。后面几行检查用户以前是否访问过这个页面（也就是，lastVisit 是否非空），如果访问过，就显示他上一次访问的时间。

5. `document.getElementById("cookieData").innerHTML = outMsg;`

最后，将 outMsg 显示在屏幕上，告诉用户以前他访问这个页面的情况。这个脚本的结果见图 9-8。

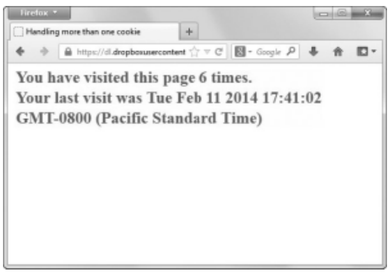


图 9-8　两个 cookie 中的信息（以及其他一些文本）被写到屏幕上

9.7　显示新内容提醒信息

可以使用 cookie 和 JavaScript 提醒经常访问站点的用户注意他们没看到过的内容。这会向访问你站点的用户提供更个性化的体验，让站点显得更智能、友好。如果通过 cookie 发现一些内容是在访问者上次访问之后添加的，那么脚本 9-9、脚本 9-10 和脚本 9-11 就在这些行的前面添加一个小的 New! 图像（见图 9-9）。同样，这里的许多代码与本章前面的示例相似。

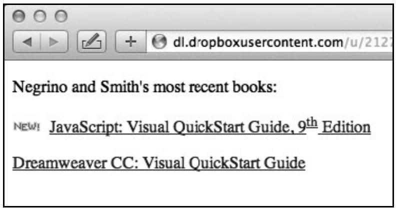


图 9-9　JavaScript 在 cookie 中查询用户上一次访问的时间，并且标出新内容

脚本 9-9　这个页面的 HTML 将下一个脚本的结果应用于页面

```
<!DOCTYPE html>
<html>
<head>
     <title>New for You</title>
     <link rel="stylesheet" href="script07.css">
     <script src="script07.js"></script>
</head>
<body>
     <p>Negrino and Smith's most recent books:</p>
     <p id="New-20140601"><a href="http://www.javascriptworld.com">JavaScript:Visual QuickStart Guide,
     ➝9<sup>th </sup> Edition</a></p>
     <p id="New-20130812"><a href="http://www.dreamweaverbook.com">Dreamweaver CC: Visual QuickStart
     ➝Guide</a></p>
</body>
</html>
```

脚本 9-10　CSS 与 JavaScript 和 HTML 相结合，可以产生个性化的内容

```
body {
     background-color: #FFF;
}

p.newImg {
     padding-left: 35px;
     background-image: url(images/new.gif);
     background-repeat: no-repeat;
}
```

脚本 9-11　这个脚本可以提醒用户注意新内容，从而使站点更加个性化

```
window.addEventListener("load",initPage,false);

function initPage() {
     var now = new Date();
     var lastVisit = new Date(cookieVal("pageVisit"));
     var expireDate = new Date();
```

```
        expireDate.setMonth(expireDate.getMonth()+6);

        document.cookie = "pageVisit=" + now + ";expires=" + expireDate.toGMTString();
        var allGrafs = document.getElementsByTagName("p");

        for (var i=0; i<allGrafs.length; i++) {
           if (allGrafs[i].id.indexOf("New-") != -1) {
              newCheck(allGrafs[i],allGrafs[i].id.substring(4));
           }
        }

        function newCheck(grafElement,dtString) {
           var yyyy = parseInt(dtString.substring(0,4),10);
           var mm = parseInt(dtString.substring(4,6),10);
           var dd = parseInt(dtString.substring(6,8),10);
           var lastChgd = new Date(yyyy,mm -1,dd);

           if (lastChgd.getTime() > lastVisit.getTime()) {
              grafElement.className += " newImg";
           }
        }
    }

    function cookieVal(cookieName) {
        var thisCookie = document.cookie.split("; ");

        for (var i=0; i<thisCookie.length; i++) {
           if (cookieName == thisCookie[i].split("=")[0]) {
              return thisCookie[i].split("=")[1];
           }
        }
        return "1 January 1970";
    }
```

9

⇛ 显示新内容提醒信息

1. ```
   <a href="http://www.javascriptworld.com">JavaScript: Visual QuickStart
   ➝Guide, 9<sup>th</sup> Edition</a>
   <a href="http://www.dreamweaverbook.com">Dreamweaver CC: Visual
   ➝QuickStart Guide</a>
   ```

在脚本 9-9 中，这两个段落上的 id 属性将所包含日期告诉 JavaScript（稍后我们将会看到），JavaScript 将这些日期与后面步骤中建立的信息进行比较。

2. ```
   p.newImg {
     padding-left: 35px;
     background-image: url(images/new.gif);
     background-repeat: no-repeat;
   }
   ```

在脚本 9-10 中，对于页面上具有 newImg 类的段落（在<p>标签中），我们使用 CSS 在左边添加 35 像素的内边距（padding），并且将 New!图像放在背景中。但是，因为内边距确保段落内容的前面是空的，所以这个图像看起来不像是背景图像，而是段落内容前面的一个标志。

3. `var lastVisit = new Date(cookieVal("pageVisit"));`

```
var expireDate = new Date();
expireDate.setMonth(expireDate.getMonth()+6);
```

在脚本 9-11 中，这些代码对 lastVisit 和 expireDate 日期进行初始化。第一个是 cookie 中保存的访问者上一次访问站点的日期，第二个是重写 cookie 时将使用的过期日期。

4. `document.cookie = "pageVisit=" + now + ";expires=" + expireDate.toGMTString();`

这一行写 cookie，它将当前日期写到 pageVisit 中，将 expireDate 的值写到 expires 中。

5. `var allGrafs = document.getElementsByTagName("p");`

这一行创建一个包含页面上所有<p>元素的数组，这样就可以遍历它们，逐一寻找我们要处理的元素。

6. `for (var i=0; i<allGrafs.length; i++) {`

这里，开始一个对数组进行遍历的循环，依次检查每个段落元素。

7. `if (allGrafs[i].id.indexOf("New-")!= -1) {`

如果这个段落具有一个包含文本 New-的 id 属性，我们就知道这是要关注的段落，所以要执行后面的操作。

8. `newCheck(allGrafs[i],allGrafs[i].id.substring(4));`

我们要检查这个段落是否是这个访问者没看过的。newCheck()函数执行这个检查，传递给它的参数有两个：当前的段落元素（allGrafs[i]）和 id 属性的第二部分。substring()从字符串中提取出第五个字符到末尾的子字符串。因为这是我们此处唯一关注的，所以只需要传递它。（请记住，JavaScript 字符串中的字符是从零开始编号的，所以字符串中的第五个字符在位置 4 上。）

9. `function newCheck(grafElement, dtString) {`

这个函数需要传递进两个参数，这两个参数在函数内部称为 grafElement（段落元素）和 dtString（id 属性的第二部分）。

10. `var yyyy = parseInt(dtString.substring(0,4),10);`

```
var mm = parseInt(dtString.substring(4,6),10);
var dd = parseInt(dtString.substring(6,8),10);
```

这三行代码从字符串中解析出日期。例如，“20140601”就是 2014 年 6 月 1 日。

yyyy 变量存储前 4 位（从第 0 位开始，到第 4 位前面结束），结果是“2014”。mm 变量存储第 4 位和第 5 位，dd 变量存储第 6 位和第 7 位。在每种情况中都对结果应用 parseInt()，从而将 substring()返回的值转换为整数。

11. `var lastChgd = new Date(yyyy,mm-1, dd);`

因为已经获得了年、月和日，所以可以设置 lastChgd。但是，等一下！JavaScript 的日期格式比较古怪，必须将月份减 1 才能得到正确的结果——请记住，只将月份减 1，对于年和日不要这么做。实际上，月份是基于零的，而年和日是基于 1 的。（关于日期及其古怪的更多信息，请参考第 11 章。）

12. `if (lastChgd.getTime() > lastVisit.getTime()) {`

现在，可以对两个日期进行比较。只有在最后修改页面信息的日期晚于访问者上次访问的日期的情况下，才执行以下操作。

13. `grafElement.className += " newImg";`

这里是核心部分：我们知道这个段落应该显示 New!图像。所以在这个<p>标签中添加一个 class

属性 newImg，HTML 页面中声明的样式就会自动应用于这个段落，因此会显示图像。

也就是说，我们可以使用 JavaScript 在元素中添加属性及其相关联的值。在这个示例中，元素是 <p>，属性是 class，属性值是 newImg。因为元素可能已经有了 class 属性，这行代码会添加属性值，而不是覆盖当前的值。

添加了这个新属性之后，会立即触发浏览器的呈现引擎，自动地将样式应用于这个元素，从而使图像出现。

14.
```
function cookieVal(cookieName) {
    var thisCookie = document.cookie.split("; ");

    for (var i=0; i<thisCookie.length; i++) {
      if (cookieName == thisCookie[i].split("=")[0]) {
         return thisCookie[i].split("=")[1];
      }
    }
    return "1 January 1970";
}
```

这是我们已经熟悉的 cookieVal()函数。这里的唯一差异是，如果没有找到名字指定的 cookie，它会返回"1 January 1970"，而不是零。这会简化其他地方的代码。这是 JavaScript 能够识别的最早的日期，所以其他日期都应该晚于这个日期。这个日期并不向用户显示，它只是 JavaScript 内部引用的日期。

✔提示

- 你可能更熟悉只向 parseInt()传递一个参数的调用方式。但是，这里传递了两个参数：要转换的字符串和 10。最后一个参数让 parseInt()总是返回十进制的数字。如果不这么做，那么在把以 0 开头的字符串传递给 parseInt()时，它可能把结果转换为八进制，这会产生错误的结果。在这种情况下，调用 parseInt("09")和 parseInt("09",10)会返回不同的结果，而后者才是我们需要的。这只是你需要了解的 JavaScript 的怪脾气之一。
- substring(to,from)命令返回一个字符串中从 to 位置开始，到 from 位置前面结束的字符（字符从零开始编号）。所以，如果字符串包含"20140807"，而你希望获得第 5 个和第 6 个字符，就要使用 substring(4,6)，结果是字符串"08"。

  from 参数是可选的。如果省略它，就表示要获得从 to 位置开始到末尾的所有字符。

9

第 10 章

对象和 DOM

W3C 建议符合标准的浏览器采用节点操纵（node manipulation）的方式支持网页，使页面表现得更像应用程序，而不是一般的标准静态页面。例如，可以让页面在不与服务器进行通信的情况下，根据用户输入的内容发生改变，以及在脚本的控制下更新页面。尽管可以像本书中其他地方所做的那样，使用 `innerHTML` 这样的技术，但是在本章中我们将讲解官方支持的方式。节点操纵也可以在服务器端实现，但是如果要提供这个功能，同时又不必迫使用户在页面之间不断移动，那么只能使用 JavaScript。

在本章中，你将学习一些关于节点和 DOM 的知识，添加、删除和操作特定的节点，以及在页面上插入和替换节点。

10.1　关于节点操纵

本章是本书中探索 JavaScript 和 DOM 最深入的一章，所以我们首先需要讨论一下历史和术语。

10.1.1　DOM 2 和W3C

W3C（见第 1 章中的介绍）发布了规范来规定浏览器应该如何处理文档对象模型（Document Object Model，DOM）。DOM Level 2 规范于 2000 年 11 月成为正式推荐标准，它更深入地规定了浏览器应该如何引用和管理页面上的内容。可以在 http://www.w3.org/TR/DOM-Level-2-Core/上找到这个规范的更多细节。

尽管这个规范已经发布好几年了，但是当前使用的许多浏览器仍然只具有不完整的 DOM 2 支持。在使用本章中的脚本之前，要确保目标受众能够运行它们，或者为老式浏览器提供实现同样效果的替代方法。好消息是，当今的大多数网上冲浪者都使用 IE 9+、Firefox、Chrome 或 Safari，而这些浏览器都能够很好地运行这些脚本。

10.1.2　DOM 2 术语

在本书的开头，我们将 JavaScript 称为“组合式（snap-together）语言”，因为可以将对象、属性和方法组合在一起来构建出 JavaScript 应用程序。还有一种看待 HTML 页面的方式：将它看作由节点（node）组成的树结构。例如，下面这个简单的 HTML 页面：

```
<html>
<head>
  <title>My page</title>
</head>
<body>
  <p>This is text on my page</p>
</body>
</html>
```

可以看作图 10-1 所示的结构。

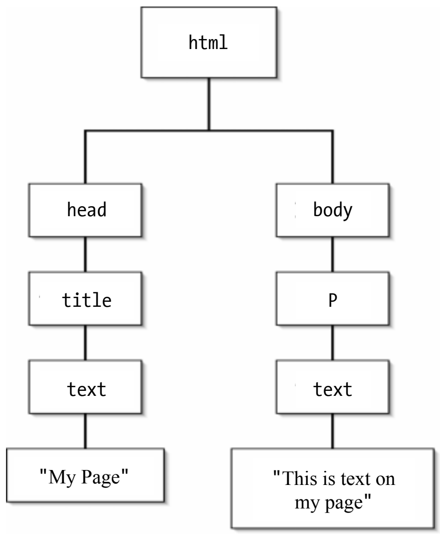


图 10-1 这个由节点组成的树结构只是看待 HTML 页面组织方式的另一种方法

可以使用 JavaScript 修改这个树结构的任何方面，包括添加、访问、修改和删除树中的节点。这个树中的每个框都是一个节点。如果一个节点包含 HTML 标签，那么它就称为*元素节点*（element node）。否则，它称为*文本节点*（text node）。当然，元素节点可以包含文本节点。

10.1.3 DOM 3

DOM Level 3 于 2004 年 4 月成为正式推荐标准。关于这个规范的信息，可以在 www.w3.org/TR/DOM-Level-3-Core/找到。与 W3C 过程的许多部分一样，仍然要再过很长时间在浏览器中才会出现真正的支持。所以本章只讨论 DOM 2。但是，如果你有兴趣了解更多情况，那么可以从 ECMAScript binding 开始（www.w3.org/TR/DOM-Level-3-Core/ecma-script-binding.html）。

10.2 添加节点

学习节点最容易的方法，是首先在文档末尾追加一个元素节点（它将包含一个文本节点）。脚本 10-1 和脚本 10-2 让用户输入一些数据并且单击一个按钮，然后将一个新段落添加到页面中（见图 10-2）。

10

脚本 10-1　这个 HTML 创建一个文本区域和一个提交按钮，让用户能够添加文本节点

```
<!DOCTYPE html>
<html>
<head>
     <title>Adding Nodes</title>
     <script src="script01.js"></script>
</head>
<body>
     <form id="theForm">
        <p><textarea id="textArea" rows="5"cols="30"></textarea></p>
        <input type="submit" value="Add some text to the page">
     </form>
</body>
</html>
```

脚本 10-2　这个脚本使用户可以将任何文本添加到页面上

```
window.addEventListener("load",initAll,false);

function initAll() {
     document.getElementById("theForm").addEventListener("submit",addNode,false);
}

function addNode(evt) {
     var inText = document.getElementById("textArea").value;
     var newText = document.createTextNode(inText);

     var newGraf = document.createElement("p");
     newGraf.appendChild(newText);

     var docBody = document.getElementsByTagName("body")[0];
     docBody.appendChild(newGraf);

     evt.preventDefault();
}
```

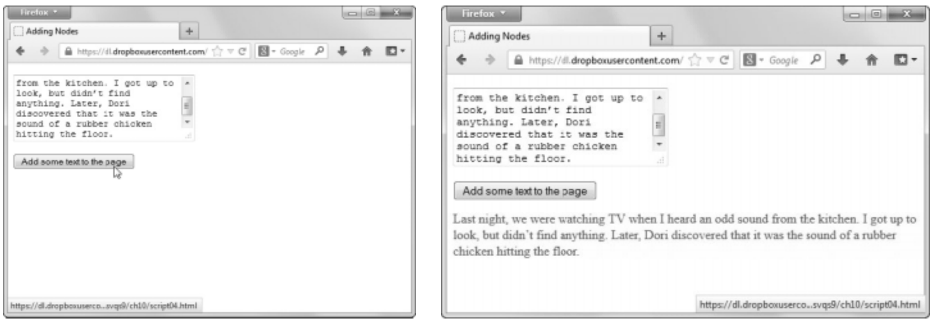


图 10-2　在字段中输入文本，然后单击按钮（左图），就可以添加一个节点。这些文本会出现在页面上（右图）

⇒ 添加节点

1. `var newText = document.createTextNode(inText);`

首先，使用 createTextNode()方法创建一个新的文本节点（名为 newText），它将包含在 textArea

中找到的文本。

2. var newGraf = document.createElement("p");

接下来，使用 createElement()方法创建一个新的元素节点。尽管这里创建的节点是一个段落标签，但它也可以是任何 HTML 容器（div、span 等）。这个新元素的名称是 newGraf。

3. newGraf.appendChild(newText);

为了将新文本添加到新段落中，我们必须调用 appendChild()。这是 newGraf 元素的一个方法，将 newText 传递给它时，就会将文本节点放进段落中。

4. var docBody = document.getElementsByTagName("body")[0];

为了将新节点添加到文档的 body 中，需要查明 body 的位置。这个 getElementsByTagName()方法会给出页面上的每个 body 标签。如果页面符合标准，那么应该只有一个 body 标签。[0]属性是第一个 body 标签，我们将它存储在 docBody 中。

5. docBody.appendChild(newGraf);

最后，将 newGraf 追加到 docBody 中（同样使用 appendChild()），从而将用户输入的新文本放到页面上。

6. evt.preventDefault();

现在使用事件处理高级方法（参见第 8 章），为了阻止表单提交我们需要采取一些措施。不过，只是返回 false 没用，我们通知调用事件不要运行任何默认事件。

✔提示

- 既然对 innerHTML 进行简单的赋值就可以实现同样的效果，那么为什么要这么费事儿（创建文本节点、创建元素节点并且追加子节点）呢？有一个原因：利用这种方式，就不可能导致页面无效。例如，添加的每个<p>或<div>标签会自动结束。另一方面，如果使用 innerHTML，就非常容易形成无效的标签（简直太容易了）。一旦发生这种情况，页面的 DOM 就很难处理了。例如，如果一个元素有开始标签，而没有结束标签，那么就无法读取这个元素的内容。
- 段落本身不能再包含段落。如果你试图传入多个用空行分隔的句子，这段代码将把它们合并成一个大段，而不是分别解析每一段。

10

10.3　删除节点

如果你希望在页面上添加内容，那么也可能希望从页面中删除内容。脚本 10-3（HTML）和脚本 10-4 删除页面上的最后一个段落，如图 10-3 所示。

脚本 10-3　这个脚本添加一个用来删除文本节点的链接（而不是按钮）

```
<!DOCTYPE html>
<html>
<head>
     <title>Deleting Nodes</title>
     <script src="script02.js"></script>
</head>
<body>
     <form id="theForm">
        <p><textarea id="textArea" rows="5" cols="30"></textarea></p>
        <input type="submit" value="Add some text to the page">
```

```
	</form>
	<a href="#"id="deleteNode" >Delete last paragraph</a>
</body>
</html>
```

脚本 10-4　现在，用户既可以添加文本，也可以删除文本

```
window.addEventListener("load",initAll,false);

function initAll() {
	document.getElementById("theForm").addEventListener("submit",addNode,false);
	document.getElementById("deleteNode").addEventListener("click",delNode,false);
}

function addNode(evt) {
	var inText = document.getElementById("textArea").value;
	var newText = document.createTextNode(inText);

	var newGraf = document.createElement("p");
	newGraf.appendChild(newText);

	var docBody = document.getElementsByTagName("body")[0];
	docBody.appendChild(newGraf);

	evt.preventDefault();
}

function delNode(evt) {
	var allGrafs = document.getElementsByTagName("p");

	if (allGrafs.length > 1) {
	   var lastGraf = allGrafs[allGrafs.length-1];
	   var docBody = document.getElementsByTagName("body")[0];
	   docBody.removeChild(lastGraf);
	}
	else {
	   alert("Nothing to remove!");
	}

	evt.preventDefault();
}
```

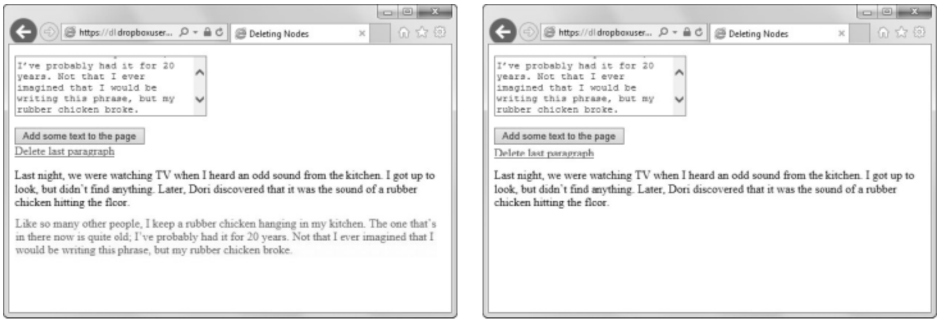


图 10-3　这个页面上的最后一个段落需要修改（左图），所以可以使用 Delete last paragraph 链接删除它（右图）

⇛ 删除节点

1. var allGrafs = document.getElementsByTagName("p");

这一行使用 getElementsByTagName 方法收集页面上的所有段落标签，并且将它们存储在 allGrafs 数组中。

2. if (allGrafs.length > 1) {

在进行实际的删除操作之前，首先必须检查 allGrafs 数组的长度是否大于 1。我们不希望尝试删除某些并不存在的东西，而且数组的长度总是至少为 1（因为脚本 10-3 中的 textarea 表单字段放在一个<p>标签中）。

3. var lastGraf = allGrafs[allGrafs.length-1];

如果有段落，那么以数组长度减 1 作为数组索引来获得最后一个段落。请记住，长度是从 1 开始的，而数组索引是从 0 开始的，所以数组长度减 1 就可以获得页面上的最后一个段落。

4. var docBody = document.getElementsByTagName("body")[0];
   docBody.removeChild(lastGraf);

与前一节中相似，为了修改文档，需要找到 body 的内容。在此之后，只需调用 docBody.removeChild() 方法并且将 lastGraf 传递给它，这告诉 JavaScript 我们希望删除哪个段落。在页面上，会立刻显示少了一个段落。

✔提示

- 同样，也可以删除段落之外的其他元素节点。为此，只需修改脚本，向 getElementsByTagName() 传递其他非 p 标签名称。

10.4　删除特定的节点

尽管总是删除最后一个段落可能是有意思的，但是有时候希望删除不在页面末尾的段落。脚本 10-5（HTML）和脚本 10-6 使我们的代码更加灵活，让用户能够选择要删除的段落，见图 10-4 和图 10-5。

脚本 10-5　单选按钮让访问者选择是添加文本，还是删除文本

```
<!DOCTYPE html>
<html>
<head>
     <title>Deleting Selected Nodes</title>
     <script src="script03.js"></script>
</head>
<body>
     <form id="theForm">
        <p><textarea id="textArea" rows="5"cols="30"></textarea></p>
        <p>
           <label><input type="radio" name="nodeAction">Add node</label>
           <label><input type="radio" name="nodeAction">Delete node</label>
        </p>
        Paragraph #: <select id="grafCount"></select>
        <input type="submit" value="Submit">
     </form>
     <div id="modifiable"> </div>
</body>
</html>
```

脚本 10-6　这个脚本让用户能够选择要删除的段落

```
window.addEventListener("load",initAll,false);
var nodeChgArea;

function initAll() {
     document.getElementById("theForm").addEventListener("submit",nodeChanger,false);
     nodeChgArea = document.getElementById("modifiable");
}

function addNode() {
     var inText = document.getElementById("textArea").value;
     var newText = document.createTextNode(inText);

     var newGraf = document.createElement("p");
     newGraf.appendChild(newText);

     nodeChgArea.appendChild(newGraf);
}
function delNode() {
     var grafChoice = document.getElementById("grafCount").selectedIndex;
     var allGrafs = nodeChgArea.getElementsByTagName("p");
     var oldGraf = allGrafs.item(grafChoice);

     nodeChgArea.removeChild(oldGraf);
}

function nodeChanger(evt)  {
     var actionType = -1;
     var pGrafCt = nodeChgArea.getElementsByTagName("p").length;
     var radioButtonSet = document.getElementById("theForm").nodeAction;

     for (var i=0; i<radioButtonSet.length; i++) {
        if (radioButtonSet[i].checked) {
           actionType = i;
        }
     }

     switch(actionType) {
        case 0:
           addNode();
           break;
        case 1:
           if (pGrafCt > 0) {
              delNode();
              break;
           }
        default:
           alert("No valid action was chosen");
     }

     document.getElementById("grafCount").options.length = 0;

     for (i=0; i<nodeChgArea.getElementsByTagName("p").length; i++) {
        document.getElementById("grafCount").options[i] = new Option(i+1);
     }

     evt.preventDefault();
}
```

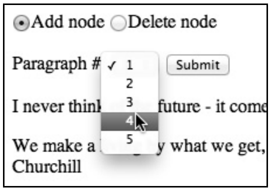


图 10-4　在添加节点之后，Paragraph #弹出菜单包含段落号的列表

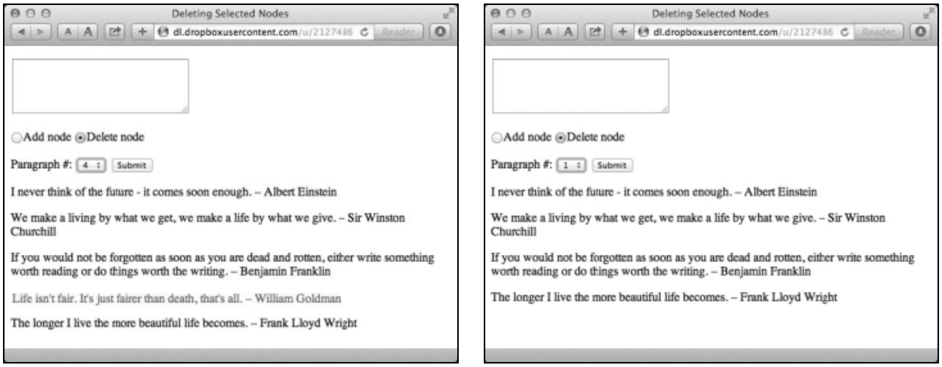


图 10-5　首先，选中 Delete node 单选按钮，然后从弹出菜单中选择要删除的段落（左图）。单击 Submit 按钮就会删除选择的段落（William Goldman 的一段话），下面的段落会向上移动（右图）

10

⇛ 删除特定节点

1. `nodeChgArea = document.getElementById("modifiable");`

因为 HTML 页面现在有多个段落，所以不容易跟踪哪些段落可以删除，哪些不可以删除。所以，我们建立一个全新的区域：一个 id 为 `modifiable` 的 `div`。在这里，将全局变量 `nodeChgArea` 设置为这个元素节点。

2.
```
var grafChoice = document.getElementById("grafCount").selectedIndex;
var allGrafs = nodeChgArea.getElementsByTagName("p");
var oldGraf = allGrafs.item (grafChoice);
```

当用户选择进行段落删除时，他们还必须选择要删除的段落。我们从 `grafCount` 字段读取段落号，并且将它存储在 `grafChoice` 中。然后，将 `allGrafs` 变量设置为 `nodeChangingArea` 中的所有段落，并且将要删除的段落存储在 `oldGraf` 中。

3. `nodeChgArea.removeChild(oldGraf);`

这个步骤与前一个示例中相同，唯一的差异是：当它运行时，我们会看到页面中间的某个段落消失了。

✔提示

- 你理解其他代码的作用有困难吗？`nodeChanger()`函数按出现的次序组合了脚本 2-15、脚本 6-5 和脚本 6-13 的功能。在程序设计中，一种非常常见的做法是，建立一个由简单例程组成的库，然后根据需要，将它们组合成更复杂的完整程序。

- 注意，在前面的代码中，我们在以前使用 docBody 的地方使用了 nodeChgArea——在操作节点时，很容易将代码改为处理其他元素节点。在这里，我们只搜索页面的一部分而不是整个页面，但是总体方法是相同的。
- 我们也可以不把 nodeChgArea 声明为全局变量并且在 initAll()中对它进行初始化，而是在使用它的每个函数中局部地创建和初始化它。这两种方式各有优缺点。在这里，我们选用全局方式，这样我们就不必多次对它进行初始化。

10.5　插入节点

除了删除不在文档末尾处的节点之外，还可能希望在文档末尾之外的其他位置添加节点。通过使用脚本 10-7（HTML）和脚本 10-8，可以选择让新节点出现在哪里。在图 10-6 中，可以看到如何插入新节点。

脚本 10-7　另一个单选按钮和一些脚本修改提供了第三个选项——在另一个段落前面插入文本

```
<!DOCTYPE html>
<html>
<head>
     <title>Inserting Nodes</title>
     <script src="script04.js"></script>
</head>
<body>
     <form id="theForm">
        <p><textarea id="textArea" rows="5" cols="30"></textarea></p>
        <p>
           <label><input type="radio" name="nodeAction">Add node</label>
           <label><input type="radio" name="nodeAction">Delete node</label>
           <label><input type="radio" name="nodeAction">Insert before node</label>
        </p>
        Paragraph #: <select id="grafCount"></select>
        <input type="submit" value="Submit">
     </form>
     <div id="modifiable"> </div>
</body>
</html>
```

脚本 10-8　用户现在可以在页面上的任何地方添加文本

```
window.addEventListener("load",initAll,false);
var nodeChgArea;

function initAll() {
     document.getElementById("theForm").addEventListener("submit",nodeChanger,false);
     nodeChgArea = document.getElementById("modifiable");
}

function addNode() {
     var inText = document.getElementById("textArea").value;
     var newText = document.createTextNode(inText);

     var newGraf = document.createElement("p");
     newGraf.appendChild(newText);

     nodeChgArea.appendChild(newGraf);
```

```
}

function delNode() {
     var grafChoice = document.getElementById("grafCount").selectedIndex;
     var allGrafs = nodeChgArea.getElementsByTagName("p");
     var oldGraf = allGrafs.item(grafChoice);

     nodeChgArea.removeChild(oldGraf);
}

function insertNode() {
     var grafChoice = document.getElementById("grafCount").selectedIndex;
     var inText = document.getElementById("textArea").value;

     var newText = document.createTextNode(inText);
     var newGraf = document.createElement("p");
     newGraf.appendChild(newText);

     var allGrafs = nodeChgArea.getElementsByTagName("p");
     var oldGraf = allGrafs.item(grafChoice);

     nodeChgArea.insertBefore(newGraf,oldGraf);
}

function nodeChanger(evt)  {
     var actionType = -1;
     var pGrafCt = nodeChgArea.getElementsByTagName("p").length;
     var radioButtonSet = document.getElementById("theForm").nodeAction;
for (var i=0; i<radioButtonSet.length; i++) {
        if (radioButtonSet[i].checked) {
           actionType = i;
        }
     }

     switch(actionType) {
        case 0:
           addNode();
           break;
        case 1:
           if (pGrafCt > 0) {
              delNode();
              break;
           }
        case 2:
           if (pGrafCt > 0) {
              insertNode();
              break;
           }
        default:
           alert("No valid action was chosen");
     }

     document.getElementById("grafCount").options.length = 0;

     for (i=0; i<nodeChgArea.getElementsByTagName("p").length; i++) {
        document.getElementById("grafCount").options[i] = new Option(i+1);
     }

     evt.preventDefault();
}
```

10

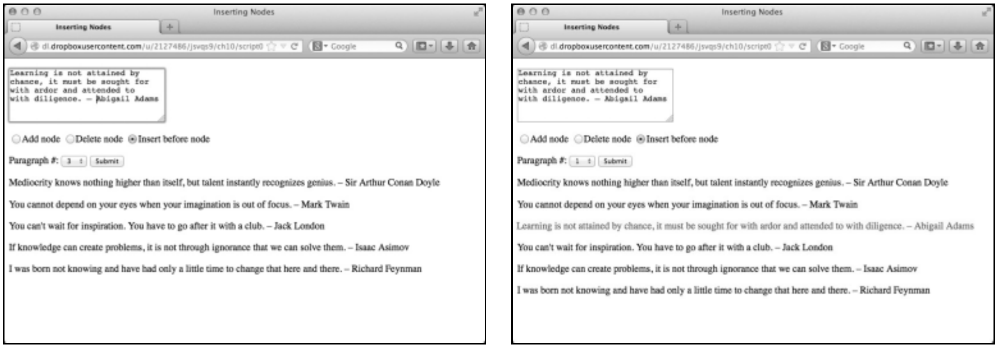


图 10-6　要想插入段落，只需单击 Insert before node 单选按钮，选择作为插入点的段落（左图），输入文本，然后单击 Submit 按钮（右图）

⇛ 插入节点

1.
```
var grafChoice = document.getElementById("grafCount").selectedIndex;
var inText = document.getElementById("textArea").value;
```

为了插入一个段落，我们需要知道两个信息：用户希望插入段落的位置（`grafChoice`）以及他们希望插入的文本（`inText`）。

2.
```
var newText = document.createTextNode(inText);
var newGraf = document.createElement("p");
newGraf.appendChild(newText);
```

这是创建新的段落节点并且以用户文本填充它的标准方法。

3.
```
var allGrafs = nodeChgArea.getElementsByTagName("p");
var oldGraf = allGrafs.item(grafChoice);
```

同样，我们收集区域中的所有 `p` 标签，然后将目标段落（我们将在这个段落前面插入新段落）存储在 `oldGraf` 中。

4.
```
nodeChgArea.insertBefore(newGraf,oldGraf);
```

插入新节点的方法是调用 `insertBefore()`方法并且传递两个参数：新节点和一个现有的节点（新节点要插入在这个节点前面）。

✔提示

- ❑ 你可能会认为，既然有 `insertBefore()`方法，就应该有 `insertAfter()`方法，但并非如此。如果希望在页面末尾添加节点，就需要使用 `appendChild()`方法。

10.6　替换节点

可以删除现有节点并且插入新节点，但是直接替换节点更简单。脚本 10-9（HTML）和脚本 10-10 演示如何用一个节点替换另一个节点。图 10-7 显示替换的效果。

脚本 10-9　这个 HTML 页面在前面示例的基础上添加了 Replace node 单选按钮

```
<!DOCTYPE html>
<html>
<head>
     <title>Replacing Nodes</title>
     <script src="script05.js"></script>
</head>
<body>
     <form id="theForm">
        <p><textarea id="textArea" rows="5" cols="30"></textarea></p>
        <p>
           <label><input type="radio" name="nodeAction">Add node</label>
           <label><input type="radio" name="nodeAction">Delete node</label>
           <label><input type="radio" name="nodeAction">Insert before node</label>
           <label><input type="radio" name= "nodeAction">Replace node</label>
        </p>
        Paragraph #: <select id="grafCount"></select>
        <input type="submit" value="Submit">
     </form>
     <div id="modifiable"> </div>
</body>
</html>
```

脚本 10-10　最终，用户可以在页面上添加、删除和替换任何文本

```
window.addEventListener("load",initAll,false);
var nodeChgArea;

function initAll() {
     document.getElementById("theForm").addEventListener("submit",nodeChanger,false);
     nodeChgArea = document.getElementById("modifiable");
}

function addNode() {
     var inText = document.getElementById("textArea").value;
     var newText = document.createTextNode(inText);

     var newGraf = document.createElement("p");
     newGraf.appendChild(newText);

     nodeChgArea.appendChild(newGraf);
}

function delNode() {
     var grafChoice = document.getElementById("grafCount").selectedIndex;
     var allGrafs = nodeChgArea.getElementsByTagName("p");
     var oldGraf = allGrafs.item(grafChoice);

     nodeChgArea.removeChild(oldGraf);
}

function insertNode() {
     var grafChoice = document.getElementById("grafCount").selectedIndex;
     var inText = document.getElementById("textArea").value;

     var newText = document.createTextNode(inText);
     var newGraf = document.createElement("p");
     newGraf.appendChild(newText);

     var allGrafs = nodeChgArea.getElementsByTagName("p");
```

```
     var oldGraf = allGrafs.item(grafChoice);

     nodeChgArea.insertBefore(newGraf,oldGraf);
}

function replaceNode() {
     var grafChoice = document.getElementById("grafCount").selectedIndex;
     var inText = document.getElementById("textArea").value;

     var newText = document.createTextNode(inText);
     var newGraf = document.createElement("p");
     newGraf.appendChild(newText);
     var allGrafs = nodeChgArea.getElementsByTagName("p");
     var oldGraf = allGrafs.item(grafChoice);

     nodeChgArea.replaceChild(newGraf,oldGraf);
}

function nodeChanger(evt)  {
     var actionType = -1;
     var pGrafCt = nodeChgArea.getElementsByTagName("p").length;
     var radioButtonSet = document.getElementById("theForm").nodeAction;

     for (var i=0; i<radioButtonSet.length; i++) {
        if (radioButtonSet[i].checked) {
            actionType = i;
        }
     }

     switch(actionType) {
        case 0:
           addNode();
           break;
        case 1:
           if (pGrafCt > 0) {
              delNode();
              break;
           }
        case 2:
           if (pGrafCt > 0) {
              insertNode();
              break;
           }
        case 3:
           if (pGrafCt > 0) {
              replaceNode();
              break
           }
        default:
           alert("No valid action was chosen");
     }

     document.getElementById("grafCount").options.length = 0;

     for (i=0; i<nodeChgArea.getElementsByTagName("p").length; i++) {
        document.getElementById("grafCount").options[i] = new Option(i+1);
     }

     evt.preventDefault();
}
```

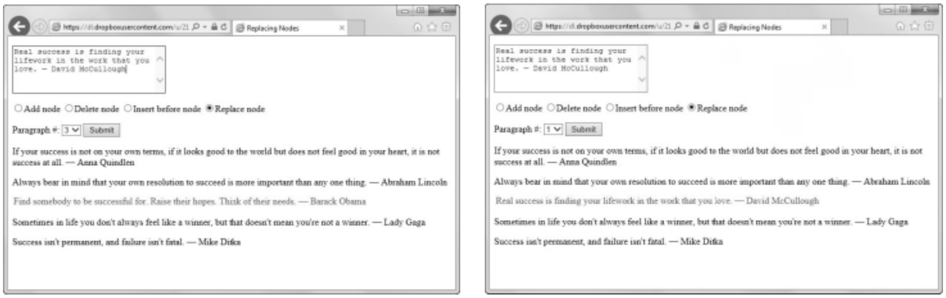


图 10-7　在这里，我们将第三个段落（左图）替换为新的文本（右图）

⇛ 替换节点

- ❑ `nodeChgArea.replaceChild(newGraf,oldGraf);`

这个脚本中只有这一行是你需要注意的新代码（对脚本其余部分的解释见本章前面的小节）。按照与前一节相似的方式，我们只需用两个参数调用 `replaceChild()`：要替换进页面的节点和要被替换掉的节点。

10.7　用对象字面量编写代码

正如后面的补充内容“关于对象字面量”所述，编写 JavaScript 的方法不只一种。脚本 10-11 演示如何把脚本 10-10 改写为使用对象字面量。

脚本 10-11　这个简短的脚本演示对象字面量的许多特性

```
window.addEventListener("load",initAll,false);

function initAll() {
     document.getElementById("theForm").addEventListener("submit",nodeChanger,false);
     chgNodes.init();
}

function nodeChanger(evt) {
     return chgNodes.doAction(evt);
}

var chgNodes = {
     actionType: function() {
        var radioButtonSet = document.getElementById("theForm").nodeAction;
        for (var i=0; i<radioButtonSet.length; i++) {
           if (radioButtonSet[i].checked) {
              return i;
           }
        }
        return -1;
     },

     allGrafs: function() {
        return this.nodeChgArea.getElementsByTagName("p");
     },

     pGrafCt: function() {
        return this.allGrafs().length;
```

```
    },

    inText: function() {
       return document.getElementById("textArea").value;
    },

    newText: function() {
       return document.createTextNode(this.inText());
    },

    grafChoice: function() {
       return document.getElementById("grafCount").selectedIndex;
    },

    newGraf: function() {
       var myNewGraf = document.createElement("p");
       myNewGraf.appendChild(this.newText());
       return myNewGraf;
    },

    oldGraf: function () {
       return this.allGrafs().item(this.grafChoice());
    },

    doAction: function(evt) {
       switch(this.actionType()) {
          case 0:
             this.nodeChgArea.appendChild(this.newGraf());
             break;
          case 1:
             if (this.pGrafCt() > 0) {
                this.nodeChgArea.removeChild(this.oldGraf());
                break;
             }
          case 2:
             if (this.pGrafCt() > 0) {
                this.nodeChgArea.insertBefore(this.newGraf(),this.oldGraf());
                break;
             }
          case 3:
             if (this.pGrafCt() > 0) {
                this.nodeChgArea.replaceChild(this.newGraf(),this.oldGraf());
                break;
             }
          default:
             alert("No valid action was chosen");
       }

       document.getElementById("grafCount").options.length = 0;

       for (var i=0; i<this.pGrafCt(); i++) {
          document.getElementById("grafCount").options[i] = new Option(i+1);
       }
       evt.preventDefault();
    },

    init: function() {
       this.nodeChgArea = document.getElementById("modifiable");
    }
}
```

关于对象字面量

到目前为止看到的 JavaScript 都是标准的过程式 JavaScript，它们采用点号表示格式：

```
var myCat = new Object;
myCat.name = "Pixel";
myCat.breed = "Tuxedo";
myCat.website = "www.pixel.mu";

function allAboutMyCat() {
       alert("Can I tell you about my cat?");
       tellMeMore = true;
}
```

如果采用对象字面量格式，这段代码应该改写为下面这样：

```
var myCat = {
       name: "Pixel",
       breed: "Tuxedo",
       website: "www.pixel.mu",
       allAbout: function() {
              alert("Can I tell you about my cat?");
              tellMeMore = true;
       }
}
```

对于这两种格式，都可以用点号表示法引用 myCat 的属性，比如 myCat.name。但是，在采用对象字面量格式时，函数变成 myCat.allAbout()而不是 allAboutMyCat()。

你可能觉得这种格式有点儿眼熟，没错，它在许多方面与 CSS 非常相似。在最基本的层面上，它实际上是一个属性—值对的列表，属性和值之间以冒号分隔，各个属性—值对之间有一个分隔符。

在使用对象字面量时，要记住几点差异。

- 使用:而不是=设置属性。
- 行以,而不是;结尾。
- 在对象中的最后一个语句上不需要逗号。

10

⇛ 使用对象字面量

1. `document.getElementsById("theForm").addEventListener("submit", nodeChanger, false);`
   `chgNodes.init();`

与前面看到的代码一样，必须首先进行初始化。第一行与以前看到的相同，但是第二行有点儿不一样：它调用 chgNodes 对象中的 init()函数。

2.
```
function nodeChanger(evt) {
    return chgNodes.doAction(evt);
}
```

这里的 nodeChanger()函数仅仅调用 chgNodes.doAction()。稍后解释为什么不直接调用这个函数。

3. `var chgNodes = {`

这是 chgNodes 对象的开头。创建对象的方式与设置变量相似，但是等号后面是包围在花括号中的一组语句。

4. actionType: function() {
```
    var radioButtonSet = document.getElementsById("theForm").nodeAction;
    for (var i=0; i<radioButtonSet.length; i++) {
      if (radioButtonSet[i].checked) {
        return i;
      }
    }
    return -1;
  },
```
在这个脚本的前一个版本中，nodeChanger()函数的第一部分设置 actionType 变量。在这里，actionType()是 chgNodes 的一个方法。尽管代码风格不一样，但最终结果应该是相同的。

5. allGrafs: function() {
```
    return this.nodeChgArea.getElementsByTagName("p");
  },

  pGrafCt: function() {
    return this.allGrafs().length;
  },
```
这是 chgNodes 中的两个简单函数：allGrafs()和 pGrafCt()。因为它们会返回值，所以可以在任何地方使用它们添加、替换或删除节点。

6. doAction: function(evt) {
```
    switch(this.actionType()) {
      case 0:
        this.nodeChgArea.appendChild(this.newGraf());
        break;
```
doAction()函数处理 chgNodes 所需的大多数工作——这几行代码只是这个函数的开头部分。与前一个版本一样，通过检查单选按钮了解应该执行哪种操作，并通过 switch 语句执行这个操作。

7. init: function() {
```
    this.nodeChgArea = document.getElementById("modifiable");
  }
```
最后是 init()函数，它的作用只是初始化 nodeChgArea 供以后使用。最重要的是，在这个例程的末尾不加逗号——除了最后一个语句之外，每个语句都应该以逗号结尾（函数本质上是一个扩展的语句）。

✔提示

- 对于步骤 1 和步骤 2，你可能会奇怪为什么不直接写成：
  `document.getElementsById("theForm")addEventListener("submit", chgNodes.doAction(), false);`
  你还可能会奇怪为什么在代码中如此频繁地使用 this？这两个问题的答案是相同的。

在对象字面量中，只需通过引用 this，就能够引用这个对象的所有属性和方法。如果使用 var 命令（就像 myNewGraf 或 radioButtonSet 那样），它就是一般的变量，不能在父对象之外访问它。如果不使用 var，而且总是以 this.XXX 形式引用它们，这些属性就成为对象本身的组成部分。

但是，对象字面量的 this 必须遵守 JavaScript 对于 this 的一般规定——this 引用的内容取决于调用它的位置。如果从表单直接调用 chgNodes.doAction()，那么 this 引用的是表单对象，这不是我们想要的结果。在 nodeChanger()中调用 chgNodes.doAction()，就能够解决这个问题。

- 如果你正在考虑放弃过程式 JavaScript，转而采用对象字面量风格，但是还没有下决心，那么请注意一个理由：脚本 10-11 的作用与脚本 10-10 完全相同，但是它的代码长度缩短了大约 20%。
- 本章没有对节点操纵进行全面的讨论，仅仅提供了一些示例来帮助你入门。如果希望进一步了解所有可用的属性和方法，那么请参考本章开头提到的 W3C 规范。

为什么要使用对象字面量

到目前为止，你主要是阅读别人的代码。如果阅读比较长的代码或者由许多人协作编写的代码，就很可能遇到看起来非常古怪的代码。这很可能是因为它们使用对象字面量，这是一种很不一样的 JavaScript 编程方式（但是同样有效）。

程序员为什么要使用对象字面量而不是过程式方法来编写 JavaScript 呢？这有以下几个原因。

- 因为每个对象（包括方法和属性）都包含在一个父对象中，所以不会遇到无意间覆盖别人的代码的问题。如果在你和你的同事各自负责的.js 文件中都有一个名为 myText 的变量，一些页面会加载这两个文件，那么页面后加载的文件优先——它会直接覆盖另一个变量，就像根本没有加载过原来的代码一样。解决方案是确保不使用全局变量，而实现这一点最简单的方法就是把你的所有代码都放在一个对象字面量中。
- 对象字面量的一个子集被称为 JavaScript Object Notation，简称为 JSON（读音与人名 Jason 近似）。JSON 是 Ajax 中最常用的数据格式之一，因此在使用 Ajax 时经常会遇到这种格式。
- 最后，与所有东西一样，编程语言的风格也在随着潮流变化。JavaScript 本身正在经历第二次（或者说第三次）飞跃，业界重新唤起了对脚本编程的兴趣。当前的潮流倾向于使用对象字面量，所以用对象字面量编写的代码会越来越多。

10

第 11 章 建立动态页面

出色的网页是许多不同因素的综合结果，包括引人注目的内容、良好的设计和对细节的关注，比如加载页面的速度有多快。加快页面加载（同时仍然向用户提供有趣的交互式体验）的方法之一，是在用户的浏览器中使用 JavaScript 对单独的页面元素进行更新。换句话说，Web 服务器并不直接向用户提供页面体验，而是通过因特网发送脚本。然后，脚本利用用户计算机的能力构造出页面。带有这种脚本的页面可以称为*动态页面*（dynamic page）。

通过将处理过程从服务器端转移到客户（用户）端，就可以获得更好的性能并且提供某种程度的个性化用户体验。

在本章中，你将学习如何使用 JavaScript 在网页上显示本地日期和时间，使用用户所在地的时间对问候语进行定制，在不同的时间格式间转换，以及在脚本的控制下跨用户的页面移动对象。

11.1 在网页上显示当前日期

JavaScript 可以判断出计算机上的当前日期和时间（以数字形式），然后以许多方式操作这个数字。但是，脚本必须处理从数字到文本型日期的转换。脚本 11-1 演示如何获得当前日期，将它从数字转换为标准的日期，然后将结果写到文档窗口。

脚本 11-1 这个脚本将当前日期写到文档窗口

```
window.addEventListener("load",initDate,false);

function initDate() {
     var dayName = new Array("Sunday","Monday", "Tuesday", "Wednesday","Thursday", "Friday", "Saturday");
     var monName = new Array("January","February", "March", "April", "May","June", "July", "August",
     →"September", "October", "November", "December");

     var now = new Date();
     var dtString = dayName[now.getDay()] + ", " + monName[now.getMonth()] + " " + now.getDate();

     document.getElementById("dtField").innerHTML = dtString;
}
```

⇒ 将当前日期写到网页

1. `window.addEventListener("load",initDate,false);`

当加载文档时，调用 `initDate()`。

2. `var dayName = new Array("Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", →"Saturday");`

首先需要创建一个新数组，其中包含一周中日的名称。一定要用逗号分隔数组中的元素，而且因为它们是文本字符串，所以每个元素必须放在引号中。将这个数组赋值给变量 dayName。

3. `var monName = new Array("January", "February", "March", "April", "May", "June", "July", →"August", "September", "October", "November", "December");`

在这一步中，创建包含月份名称的数组，并且将这个数组赋值给变量 monName。

4. `var now = new Date();`

然后，让 JavaScript 创建一个新的 Date 对象，将它命名为 now，其中包含当前日期。

5. `var dtString = dayName[now.getDay()] + "," + monName[now.getMonth()] +" " + now.getDate();`

以从右到左的次序理解 dayName[now.getDay()]对象。getDay()是用来获得一周中日的 JavaScript 方法，它向 now 对象询问今天是星期几。然后，用数字结果引用 dayName 数组中的对应元素。

接下来，在正在构造的文本字符串后加上一个逗号和一个空格，再连接下一个表达式 monName[now.getMonth()]，这个表达式返回月份名称。与前一个表达式一样，它先获得月份号，然后引用 monName 数组中的对应元素。

接下来，加一个空格，最后加上 now.getDate()，这个方法返回月中的日。将整个字符串赋值给 dtString 变量。

6. `document.getElementById("dtField").innerHTML = dtString;`

HTML 页面上的一个<span>标签具有 id dtField（HTML 非常简单，所以这里没有给出），如下所示：

```
<h1>Today is <span id="dtField"> </span>.</h1>
```

JavaScript 将 dtField 元素的 innerHTML 属性设置为 dtString 的值。结果见图 11-1。

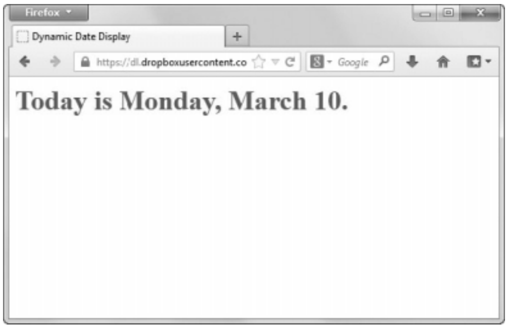


图 11-1　JavaScript 动态地在窗口中显示当前日期

JavaScript 对时间的处理方式不一致

正如本书前面提到的，JavaScript 在大多数情况下从零开始编号，所以数字序列是 0、1、2、3，等等。但是，日期是从 1 开始编号的。所以，如果有一个包含周中日的数组，就会得到：

```
Sunday = 0
Monday = 1
Tuesday = 2
Wednesday = 3
```

```
Thursday = 4
Friday = 5
Saturday = 6
```

同样，12 个月份的编号是从 0 到 11。

另一方面，在处理月中的日期时，从 0 开始编号是没有意义的（比如，不可能说 4 月 0 日），所以 JavaScript 对日期从 1 开始编号。

小时是从 0（午夜）到 23（晚上 11 点），这与 24 小时制是一致的。在本章后面，我们会演示如何把 24 小时制转换为 12 小时制。

11.2　处理周中的日期

如果某一天是周末，你可能希望向用户显示不同的消息。脚本 11-2 演示了具体做法。

脚本 11-2　这个脚本判断某一天是否是周末

```
window.addEventListener("load",initDate,false);

function initDate() {
     var now = new Date();

     if (now.getDay() > 0 && now.getDay() < 6) {
        var dtString = "Sorry, it's a weekday.";
     }
     else {
        var dtString = "Hooray, it's a weekend!";
     }

     document.getElementById("dtField").innerHTML = dtString;
}
```

⇒ 判断是否是周末

1. `var now = new Date();`

首先将变量 now 设置为当前日期。

2. `if (now.getDay() > 0 && now.getDay() < 6) {`

这一行从 now 变量中提取出数字式的周中的日期，检查它是否大于 0（记住星期日是 0）。接下来，使用&&操作符（这是逻辑与，表示两部分必须都为真），然后检查它是否小于 6（星期六）。

3. `var dtString = "Sorry, it's a weekday.";`

如果表达式的结果大于 0 且小于 6，那么它一定在 1 到 5 之间，即从星期一到星期五，所以脚本在 dtString 中设置对应的字符串。

4.
```
else {
    var dtString = "Hooray, it's a weekend!";
```

如果步骤 2 中的测试失败了，那么一定是周末，所以在 dtString 中设置对应的字符串。

5. `document.getElementById("dtField").innerHTML = dtString;`

最后，像前面的示例中一样，将 dtField 元素的 innerHTML 属性设置为 dtString 的值。结果见图 11-2。

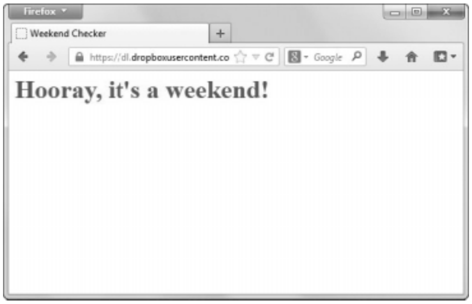


图 11-2　消息写到了窗口中

11.3　根据时间对消息进行定制

可以使用前一个示例中的技术，根据时间对消息进行定制。当用户进入站点时，这种技术常常用来提供友好的问候语。脚本 11-3 演示具体做法，图 11-3 显示的是在一般人已经入睡的时间段，页面上显示的消息。

脚本 11-3　这个脚本用来检查时间，并且作出相应的反应

```
window.addEventListener("load",initDate,false);

function initDate() {
    var now = new Date();
    document.getElementById("dtField").innerHTML = timeString(now.getHours());

    function timeString(theHour) {
       if (theHour < 5) {
          return "What are you doing up so late?";
       }
       if (theHour < 9) {
          return "Good Morning!";
       }
       if (theHour < 17) {
          return "No surfing during working hours!";
       }
       return "Good Evening!";
    }
}
```



图 11-3　如果出现这条消息，那么肯定夜深了

11

⇒ 根据时间对消息进行定制

```
❑ if (theHour < 5) {
     return "What are you doing up so late?";
```

这个脚本中的新代码首先进行一个条件测试。getHours()方法从 now 变量中提取出小时，然后检查这个数字是否小于 5（对应于凌晨 5 点，因为 JavaScript 中的小时是从午夜 0 点开始编号的）。

如果当前时间在凌晨 5 点之前，脚本就将这个消息写到文档窗口，如图 11-3 所示。

脚本的其余部分进行类似的检查，针对不同的时间，写出不同的消息。如果在凌晨 5 点和上午 9 点之间，就显示“Good Morning!”；在上午 9 点和下午 5 点之间，就显示“No surfing during working hours!”；在下午 5 点之后，就显示“Good Evening!”。

11.4　根据时区显示日期

在默认情况下，显示的日期和时间是用户计算机上的日期和时间（假设它们的设置是正确的）。如果希望显示其他地方的日期，就需要根据协调世界时（Coordinated Universal Time，UTC）计算它。UTC 本质上是格林尼治标准时间（Greenwich Mean Time，GMT）的别名，UTC 也称为“世界时间”（Universal Time，UT）。脚本 11-4 是这个页面的 HTML，脚本 11-5 中的 JavaScript 演示如何计算其他时区中的日期。

脚本 11-4　时区脚本的 HTML 使用类来标出不同办公室的时区

```
<!DOCTYPE html>
<html>
<head>
     <title>Time Zones</title>
     <script src="script04.js"></script>
</head>
<body>
     <h3>Our office hours are 9:00 am to 5:00 pm, Monday through Friday, at each of our locations.It is now
     →</h3><ul>
     <li><span class="tz-8"> </span> in San Francisco</li>
     <li><span class="tz-5"> </span> in New York</li>
     <li><span class="tz-0"> </span> in London</li>
     <li><span class="tz+7"> </span> in Hong Kong</li></ul>
</body>
</html>
```

脚本 11-5　可以调整这个脚本来显示任何时区中的时间

```
window.addEventListener("load",initDate,false);

function initDate() {
     var spanTags = document.getElementsByTagName("span");

     for (var i=0; i<spanTags.length; i++) {
        if (spanTags[i].className.indexOf("tz") == 0) {
           showTheTime(spanTags[i], spanTags[i].className.substring(2));
        }
     }
}

function showTheTime(currElem,tzOffset) {
     var dayName = new Array("Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday");
```

```
    var thatTZ = new Date();
    var dateStr = thatTZ.toUTCString();

    dateStr = dateStr.substr(0,dateStr.length - 3);
    thatTZ.setTime(Date.parse(dateStr));
    thatTZ.setHours(thatTZ.getHours() + parseInt(tzOffset));

    currElem.innerHTML = showTheHours(thatTZ.getHours()) + showZeroFilled(thatTZ.getMinutes()) + showAmPm
    →(thatTZ.getHours()) + dayName[thatTZ.getDay()];

    function showTheHours(theHour) {
       if (theHour == 0) {
          return 12;
       }
       if (theHour < 13) {
          return theHour;
       }
       return theHour-12;
    }

    function showZeroFilled(inValue) {
       if (inValue > 9) {
          return ":" + inValue;
       }
       return ":0" + inValue;
    }

    function showAmPm(thatTime) {
       if (thatTime < 12) {
          return " AM ";
       }
       return " PM ";
    }
}
```

11

⇛ 根据时区显示日期

1. `var spanTags = document.getElementsByTagName("span");`

在 initDate()函数中创建 spanTags 变量。document.getElementsByTagName("span")命令是一种很方便的技巧——span 让 JavaScript 返回一个包含页面上所有标签的数组。然后就可以遍历这个数组，寻找感兴趣的标签。

2.
```
for (var i=0; i<spanTags.length; i++) {
   if (spanTags[i].className.indexOf("tz") == 0) {
     showTheTime(spanTags[i], spanTags[i].className.substring(2));
   }
 }
```

开始一个循环，对 spanTags 数组包含的页面元素进行遍历。spanTags[i].className.indexOf ("tz")==0 的意思是，“第 *i* 个标签是否具有以 tz 开头的 class 属性”——如果是这样，就调用 showTheTime()。

传递给 showTheTime()两个参数：第一个是第 *i* 个标签元素，第二个是 class 属性（见脚本 11-4）中“tz”后面的部分，这由 substring(2)表示。为什么不在 showTheTime()函数中进行这个计算呢？这会使 showTheTime()更简单，因为第二个参数明确地提供时区偏移量。

3. function showTheTime(currElem, tzOffset) {

这个函数接受前一步中传递给它的两个参数。在这个函数中，这两个参数分别称为 currElem 和 tzOffset。

4. var thatTZ = new Date();
   var dateStr = thatTZ.toUTCString();

创建一个新的日期变量 thatTZ。下一行将日期和时间（基于 UT 形式）转换为字符串（见本章末尾的表 11-1），将结果存储在 dateStr 中。

5. dateStr = dateStr.substr(0,dateStr.length - 3);

在本节中，我们希望将 thatTZ 重新设置为基于 UT 而不是本地时间，然后就能够加上传递进来的偏移量以获得所需的结果。但是，JavaScript 不允许这么简单地处理。我们现在有了字符串格式的世界时间，但是，如果根据它重新设置时间，它会误解我们的意图，认为我们真的需要本地时间。我们需要做的是，获得日期和时间的字符串版本，并且去掉最后三个字符（即 UTC）。

6. thatTZ.setTime(Date.parse(dateStr));

去掉最后三个字符之后，可以使用 parse 方法将日期转换为毫秒，然后用 setTime 方法将 thatTZ 设置为我们需要的时间。

7. thatTZ.setHours(thatTZ.getHours() + parseInt(tzOffset));

既然已经存储了 UT 日期，就需要加上传递进来的小时数，也就是我们需要的时间与 UT 的偏移量。因为时区可以从+12 到–12，所以传递进来的时区可以是"-12"到"+12"之间的某个数。我们使用 parseInt()将这个字符串转换为–12 ~ 12 的一个数字，然后将它与当前 UT 时间相加。结果就是我们需要的值：这个时区中的正确日期和时间。

8. currElem.innerHTML = showTheHours(thatTZ.getHours()) + showZeroFilled(thatTZ.getMinutes()) +
   →showAmPm(thatTZ.getHours()) + dayName[thatTZ.getDay()];

这一行看起来让人糊涂，其实它仅仅是将其他函数的结果拼接起来，构造出要放进文档的时间值，然后设置 currElem 的 innerHTML 属性，从而将计算的结果放进文档中。最终的结果见图 11-4。

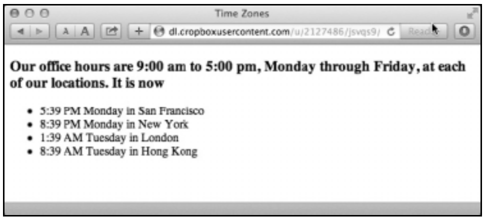


图 11-4　这个脚本根据时区计算出每个办公室的时间

下面三个函数 showTheHours()、showZeroFilled()和 showAmPm()都在 showThe Time()函数内部，所以它们可以共享变量。可以看到，这里并没有这样做，但下一节就会用到。

```
9. function showTheHours(theHour) {
     if (theHour == 0) {
        return 12;
     }
```

首先，建立 showTheHours()函数，它包含参数 theHour。如果 theHour 是零，那么返回 12（表示中午 12 点）；否则继续执行后面的代码。

10.
```
if (theHour < 13) {
  return theHour;
}
return theHour-12;
```

如果时间的小时部分小于 13，那么只需直接返回变量 theHour。否则，返回 theHour 减 12（这会将中午以后的小时数转换为 12 小时制）。

11.
```
function showZeroFilled(inValue) {
  if (inValue > 9) {
    return ":" + inValue;
  }
  return ":0" + inValue;
}
```

这个函数用来使输出更整齐。当分钟数或秒数小于等于 9 时，它在数字前面加上一个零。

12.
```
function showAmPm(thatTime) {
  if (thatTime < 12) {
    return " AM ";
  }
  return " PM ";
}
```

这个函数在时间后面添加 AM 或 PM。如果传递的变量 thatTime 小于 12，那么函数的返回值是 AM；否则，返回 PM。注意，在 AM 和 PM 文本字符串前面加上空格，这是为了让效果更好看。

✔提示

- 没有处理夏令时（Daylight Saving Time）的简便方法。一些浏览器会错误地处理夏令时。而且，是否能够处理夏令时还依赖于计算机用户是否知道如何设置他们的计算机。好在 Windows 和 OS X 能够根据因特网时间服务器自动地设置时间，这会将夏令时考虑进去，所以可以缓解这个问题。糟糕的是，JavaScript 无法从操作系统获得关于夏令时的信息，所以它无法判断出你所处的时间和地点是否应用夏令时。
- 很容易在 HTML 中添加另一个城市，而且不需要修改 JavaScript，脚本仍然可以正常工作。

11.5　把 24 小时制转换为 12 小时制

JavaScript 以 24 小时制提供时间。许多人不熟悉或不喜欢这种格式，所以可能希望将它转换为 12 小时制。在下面的两个脚本中，你将看到完成此任务的一种方法。这个页面（HTML 见脚本 11-6，CSS 见脚本 11-7）有两个重要的元素：h2 标签和两个单选按钮。脚本将时间写入前者，而使用两个单选按钮，用户可以在 24 小时制和 12 小时制之间进行切换（见图 11-5）。JavaScript 见脚本 11-8。

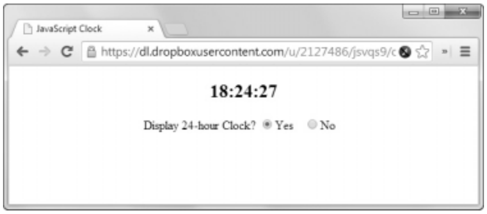


图 11-5　脚本运行的效果

脚本 11-6　这个 HTML 使用 id 标识每个单选按钮

```
<!DOCTYPE html>
<html>
<head>
     <title>JavaScript Clock</title>
     <link href="script05.css" rel="stylesheet">
     <script src="script05.js"></script>
</head>
<body>
<div class="centered">
     <h2 id="showTime"> </h2>
     Display 24-hour Clock?
     <input type="radio" name="timeClock" id="show24" checked><label for="show24">Yes</label>
     <input type="radio" name="timeClock" id="show12"><label for="show12">No</label>
</div>
</body>
</html>
```

脚本 11-7　下面的样式使页面更好看

```
body {
     background-color: #FFF;
}

.centered {
     text-align: center;
}

label {
     padding-right: 10px;
}
```

脚本 11-8　这个脚本在 24 小时制和 12 小时制之间进行转换

```
window.addEventListener("load",showTheTime,false);

function showTheTime() {
     var now = new Date();

     document.getElementById("showTime").innerHTML = showTheHours(now.getHours()) + showZeroFilled(now.
     →getMinutes()) + showZeroFilled(now.getSeconds()) + showAmPm();
     setTimeout(showTheTime,1000);

     function showTheHours(theHour) {
        if (show24Hour() || (theHour > 0 && theHour < 13)) {
           return theHour;
        }
        if (theHour == 0) {
```

```
            return 12;
        }
        return theHour-12;
    }

    function showZeroFilled(inValue) {
        if (inValue > 9) {
            return ":" + inValue;
        }
        return ":0" + inValue;
    }

    function show24Hour() {
        return (document.getElementById("show24").checked);
    }

    function showAmPm() {
        if (show24Hour()) {
            return "";
        }
        if (now.getHours() < 12) {
            return " AM";
        }
        return " PM";
    }
}
```

⇛ 把 24 小时制转换为 12 小时制

1. `document.getElementById("showTime").innerHTML = showTheHours(now.getHours()) + showZeroFilled(now.getMinutes()) + showZeroFilled(now.getSeconds()) + showAmPm();`

与前一个示例中一样，这一行看起来让人糊涂，其实它仅仅是将其他函数的结果连接起来，构造出要在页面上显示的时间值。然后将结果放进 showTime 的 innerHTML 属性。

2. `setTimeout(showTheTime,1000);`

这行代码让浏览器每秒更新一次显示。

3. `function showTheHours(theHour) {`

接下来，建立 showTheHours 函数，它包含参数 theHour。

4.
```
if (show24Hour() || (theHour > 0 && theHour < 13)) {
   return theHour;
 }
 if (theHour == 0) {
   return 12;
 }
 return theHour-12;
```

这些条件语句的意思是：如果用户希望显示 24 小时制，或者时间的小时部分大于零且小于 13（记住，||操作符表示逻辑或，你在第 1 章已经看到过它了），那么直接返回变量 theHour。如果 theHour 是零，就返回 12（表示中午 12 点）；否则，返回 theHour 减 12（这会将中午以后的小时数转换为用 12 小时制表示）。

5. function show24Hour() {

```
return document.getElementById("show24").checked;
```

这个函数根据用户在页面上选中的单选按钮，返回一个值。如果选中了 show 24，就返回 true 值，否则返回 false。

6. if (show24Hour()) {

```
    return "";
  }
  if (now.getHours() < 12) {
    return " AM";
  }
  return " PM";
```

这个函数在 12 小时时间后面加上 AM 或 PM。如果函数 show24Hour 返回 true，它就不返回任何内容，而进入下一个函数。在函数 showMilitaryTime 返回 false 的情况下，如果 now 变量的小时部分小于 12，那么这个函数返回 AM，否则返回 PM。同样，在 AM 和 PM 前面加一个空格以使输出更美观。

11.6　创建倒数计数器

有时候，可能希望在页面上建立一个倒数计数器，告诉用户离未来的某个事件还有多少天或多少小时。脚本 11-9（HTML）和脚本 11-10（JavaScript）提醒本书的另一位作者：快到我的生日和圣诞节了，该为我准备礼物了。其效果见图 11-6。

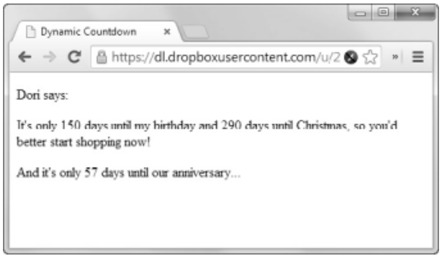


图 11-6　这个页面用于提醒另一位作者

脚本 11-9　倒数计数器脚本的 HTML

```
<!DOCTYPE html>
<html>
<head>
     <title>Dynamic Countdown</title>
     <script src="script06.js"></script>
</head>
<body>
     <p>Dori says:</p>
     <p>It's only <span class="daysTill" id="bday"> </span> days until my birthday and <span class="daysTill"
     ➝id="xmas"> </span> days until Christmas, so you'd better start shopping now!</p>
     <p>And it's only <span class="daysTill" id="anniv"> </span> days until our anniversary...</p>
</body>
</html>
```

脚本 11-10　这个脚本实现倒数计数器，它计算 Tom 还有多少天不在家

```
window.addEventListener("load",showDays,false);

function showDays() {
     var spanTags = document.getElementsByTagName("span");

     for (var i=0; i<spanTags.length; i++) {
        if (spanTags[i].className.indexOf("daysTill") > -1) {
           spanTags[i].innerHTML = showTheDaysTill(spanTags[i].id);
        }
     }

     function showTheDaysTill(thisDate) {
        var theDays;

        switch(thisDate) {
           case "anniv":
              theDays = daysTill(5,6);
              break;
           case "bday":
              theDays = daysTill(8,7);
              break;
           case "xmas":
              theDays = daysTill(12,25);
              break;
           default:
        }
        return theDays + " ";
     }

     function daysTill(mm,dd) {
        var now = new Date();
        var inDate = new Date(now.getFullYear(),mm-1,dd);

        if (inDate.getTime() < now.getTime()) {
           inDate.setYear(now.getFullYear()+1);
        }

        return Math.ceil(dayToDays(inDate) - dayToDays(now));
     }

     function dayToDays(inTime) {
        return inTime.getTime() / (1000 * 60 * 60 * 24);
     }
}
```

⇛ 创建倒数计数器

1. `var spanTags = document.getElementsByTagName("span");`

创建一个新数组 spanTags，并用页面上的所有标签填充它。

2.
```
for (var i=0; i<spanTags.length; i++) {
   if (spanTags[i].className.indexOf("daysTill") > -1) {
     spanTags[i].innerHTML = showTheDaysTill(spanTags[i].id);
   }
```

这个循环遍历 spanTags 数组，检查在页面上任何标签的 class 属性中是否能够找到字符串 daysTill。

11

请记住，一个标签可以有多个 class 属性（即 class="firstClass daysTill somethingElse fourthThing"）。

如果找到了 daysTill，就调用 showTheDaysTill()函数，并且传递给它一个参数：这个标签的 id（这个 id 表示要在这个页面中显示哪个日期）。这个函数返回一个值，然后将这个值放进 innerHTML。

3.
```
switch(thisDate) {
    case "anniv":
      theDays = daysTill(5,6);
      break;
    case "bday":
      theDays = daysTill(8,7);
      break;
    case "xmas":
      theDays = daysTill(12,25);
      break;
    default:
```

如果你不记得 switch/case 多级条件结构了，那么可以回顾第 2 章中的讨论。在这里，我们使用 thisDate 的值针对 3 个 case 语句进行测试。对于 anniv 分支，将 theDays 设置为 5 月 6 日（5、6 是数字表示，很像现实中日期的写法）；对于 bday，将它设置为 8 月 7 日；对于 xmas，theDays 设置为 12 月 25 日。

4.
```
return theDays + " ";
```

showTheDays()函数最后返回日的编号和一个空格。加空格是为了解决 IE 中的一个问题：它会消除 HTML 中的空格。如果脚本不在末尾返回空格，数字就会与单词 days 靠在一起。如果把单词 days 放在这个函数中，就不需要在后面加空格了。

5.
```
function daysTill(mm,dd) {
    var now = new Date();
    var inDate = new Date(nowgetFullYear(),mm-1,dd);
```

这一步建立 daysTill()函数，这个函数接受来自步骤 3 中的 case 语句的日期。然后，创建变量 now 和 inDate。后一个变量设置为当前年份以及传递进来的月份（要减 1 才能获得正确的结果，见后面的补充内容“时间处理的其他怪异之处”）和日编号。

6.
```
if (inDate.getTime() < now.getTime()) {
    inDate.setYear (now.getFullYear()+1);
  }
```

然后，将这个日期与今天进行比较。如果今年的这个日期已经过了，就将年份递增，从而以下一年的日期作为对比的目标。

7.
```
return Math.ceil(dayToDays(inDate) - dayToDays(now));
```

在这里，我们计算 inDate 和当前日期之间的天数。Math.ceil()方法确保结果是一个整数。

8.
```
function dayToDays(inTime) {
    return inTime.getTime() / (1000 * 60 * 60 * 24);
```

JavaScript 将日期存储为自 1970 年 1 月 1 日以来的毫秒数。为了比较两个日期，需要把它改为自 1970 年 1 月 1 日以来的天数。首先，计算一天的毫秒数：1000（1 秒内的毫秒数）乘以 60（1 分钟内的秒数），

再乘以 60（1 小时内的分钟数），然后乘以 24（1 天内的小时数）。将 getTime()返回的毫秒数除以这个数字，就会得到自 1970 年 1 月 1 日以来的天数。

时间处理的其他怪异之处

在 JavaScript 中，月份从 0 开始编号，而日从 1 开始编号，而且根据浏览器使用的 JavaScript 版本，JavaScript 处理 1970 年之前的年份的方法也不一致。

Navigator 2（使用 JavaScript 1.0）根本不能处理 1970 年之前的年份，而且有 2000 年问题（Year 2000 Problem），因此对于 2000 年或之后的日期，会返回错误的结果。在 Navigator 3（使用 JavaScript 1.1）中，如果年份在 20 世纪，getYear()方法返回的值是两位的；如果年份在 1900 年之前或 2000 年之后，就返回四位数字。但是，并非所有 Netscape 版本都是这种情况，例如 Mac 的 Netscape Navigator 4 对于 2000 年返回 100。更糟糕的是，在 Firefox 中仍然有这种情况——在当前版本（27）中，getYear()仍然返回两位数字（对于 21 世纪的年份就会出现错误）。

JavaScript 1.2（在 Navigator 4 以及兼容 ECMAScript 的浏览器中使用，比如 IE 4 和更高版本）引入了一个新方法 getFullYear()，这个方法总是返回四位的年份。如果你为版本 4 和更高版本的浏览器设计页面，那么建议使用 getFullYear()，我们在本书中就是这么做的。

由于某些原因，JavaScript 中的 getTime()方法返回自 1970 年 1 月 1 日以来的毫秒数。好在我们几乎用不着查看这个数字，因为过去 40 年的毫秒数是一个非常大的数字。

11.7　隐藏和显示层

尽管 HTML、CSS 和 JavaScript 组合可以形成一个文档，但是有时候让效果看起来像是有多个文档更有用。例如，通过 CSS 和 JavaScript 的组合，在当前 HTML 页面内部（或上面）显示弹出窗口之类的东西。不，这不使用废弃的 Netscape layer 标签，它只是看起来像是单独的一层，用户只关心效果。

这需要 3 个文档：HTML 文档（见脚本 11-11）、CSS 样式表（见脚本 11-12）和 JavaScript 文件（见脚本 11-13）。我们通过 JavaScript 使用 HTML 中分配的 id 来操作图像，使用 CSS 设置广告在页面上的位置，尤其是它的 z-index，这个数字决定哪个对象显示在其他对象上面。当两个对象占据同一空间时，会显示 z-index 值大的对象。

脚本 11-11　这个广告示例的 HTML 使用 id 标出要操作的元素

```
<!DOCTYPE html>
<html>
<head>
     <title>Layered Divs</title>
     <link href="script07.css" rel="stylesheet">
     <script src="script07.js"></script>
</head>
<body>
     <div id="annoyingAdvert">
        This is an incredibly annoying ad of the type you might find on some web sites.
        <div id="closeBox">&otimes;</div>
     </div>
     <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean lacus elit, volutpat vitae,
```

```
    →egestas in, tristique ut, nibh. Donec congue lacinia magna. Duis tortor justo, dapibus vel,
    →vulputate sed, mattis sit amet, leo. Cras purus quam, semper quis, dignissim id, hendrerit eget,
    →ante. Nulla id lacus eget nulla bibendum venenatis. Duis faucibus adipiscing mauris. Integer
    →augue. In vulputate purus eget enim. Nam odio eros, porta vitae, bibendum sit amet, iaculis nec,
    →elit. Cras egestas scelerisque pede. Donec a tellus. Nullam consectetuer fringilla nunc.</p>

    <p>Nam varius metus congue ligula. In hac habitasse platea dictumst. In ut ipsum a pede rhoncus
    →convallis.Sed at enim. Integer sed metus quis est egestas vestibulum. Quisque mattis tortor
    →a lorem. Nam diam. Integer consequat lectus. Donec molestie elementum nisl. Donec ligula sapien,
    →volutpat eget, dictum quis, mollis a, odio. Aliquam augue enim, gravida nec, tempor ac, interdum
    →in, urna. Aliquam mauris. Duis massa urna, ultricies id, condimentum ac, gravida nec, dolor.
    →Morbi et est quis enim gravida nonummy. Cum sociis natoque penatibus et magnis dis parturient
    →montes, nascetur ridiculus mus. Mauris nisl quam, tincidunt ultrices, malesuada eget, posuere
    →eu, lectus. Nulla a arcu. Sed consectetuer arcu et velit. Quisque dignissim risus vel elit.</p>

    <p>Nunc massa mauris, dictum id, suscipit non, accumsan et, lorem. Suspendisse non lorem quis dui
    →rutrum vestibulum. Quisque mauris. Curabitur auctor nibh non enim. Praesent tempor aliquam
    →ligula. Fusce eu purus. Vivamus ac enim eget urna pulvinar bibendum. Integer porttitor, augue
    →et auctor volutpat, lectus dolor sagittis ipsum, sed posuere lacus pede eget wisi. Proin vel
    →arcu ac velit porttitor pellentesque. Maecenas mattis velit scelerisque tellus. Cras eu tellus
    →quis sapien malesuada porta. Nunc nulla. Nullam dapibus malesuada lorem. Duis eleifend rutrum
    →tellus. In tempor tristique neque. Mauris rhoncus. Aliquam purus.</p>

    <p>Morbi felis quam, placerat sed, gravida a, bibendum a, mauris. Aliquam porta diam. Nam consequat feugiat
    →diam. Fusce luctus, felis ut gravida mattis, ante mi viverra sapien, a vestibulum tellus lectus
    →ut massa. Duis placerat. Aliquam molestie tellus. Suspendisse potenti. Fusce aliquet tellus
    →a lectus. Proin augue diam, sollicitudin eget, hendrerit non, semper at, arcu. Sed suscipit
    →tincidunt nibh. Donec ullamcorper. Nullam faucibus euismod augue. Cras lacinia. Aenean
    →scelerisque, lorem sed gravida varius, nunc tortor gravida odio, sed sollicitudin pede augue
    →ut metus. Maecenas condimentum ipsum et enim. Sed nulla. Ut neque elit, varius a, blandit quis,
    →facilisis sed, velit. Suspendisse aliquam odio sed nibh.</p>
</body>
</html>
```

脚本 11-12　CSS 对这个层应用样式，让它看起来与文档的其余部分不一样

```
body {
    background-color: #FFF;
}

#annoyingAdvert {
    position: absolute;
    z-index: 2;
    display: none;
    width: 100px;
    background-color: #FFC;
    padding: 10px;
    margin: 10px;
    border: 5px solid yellow;
}

#closeBox {
    position: absolute;
    color: red;
    font-size: 1.5em;
    top: 0;
    right: 0;
}
```

脚本 11-13　JavaScript 显示这个层并允许用户隐藏它（谢天谢地）

```
window.addEventListener("load",initAdvert,false);

function initAdvert() {
     var adBox = "annoyingAdvert";

     document.getElementById(adBox).style.display = "block";
     document.getElementById("closeBox").addEventListener
         "click"
         function() {
            document.getElementById(adBox).style.display = "none";
         },
         false
   );
}
```

⇛ 隐藏和显示对象

1. var adBox = "annoyingAdvert";

   document.getElementById(adBox).style.display = "block";

看了脚本 11-11，就会知道要显示的层的 id 是 annoyingAdvert。脚本 11-11 告诉这个层它最初应该是隐藏的，所以看不到它。但是，在加载页面之后，脚本通过把 display 属性设置为 block 来显示它。

2. document.getElementById("closeBox").addEventListener("click")

```
    function() {
      document.getElementById(adBox).style.display = "none";
    },
    false
  );
```

为什么这个元素名为 annoyingAdvert？因为无法阅读它背后的内容（见图 11-7）。但是，我们不想讨人厌，所以让用户能够通过单击关闭控件关闭这个层（即隐藏它）。把 display 属性设置为 none，就会关闭这个层。

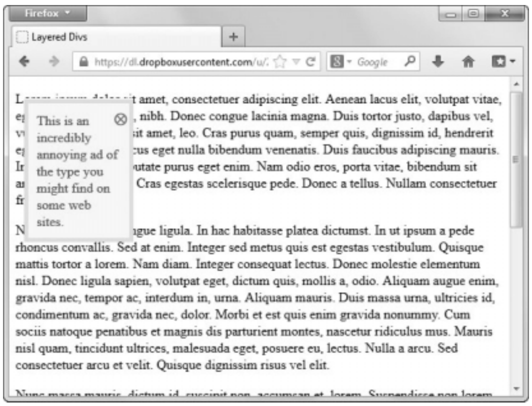


图 11-7　广告出现在左边，用户可以关闭它

11.8　移动文档中的对象

可以使用 JavaScript 让对象（图像、文本等）在屏幕上到处移动。实际上，甚至可以让对象看起来像是在进行三维移动，让对象在文档中的其他对象背后移动。在这个示例中，你将看到前个任务中的广告图像是如何更令人讨厌的。

这个示例需要 3 个文档，其中 HTML 文档和 CSS 样式表与前一版本一样，这里仅给出 JavaScript 文件（见脚本 11-14）。现在，当用户要关闭广告层时，它会“跑开”。谢天谢地，在跑出屏幕之前它会停下来（见图 11-8），最终能够关闭它！

脚本 11-14　移动广告的 JavaScript

```
window.addEventListener("load",initAdvert,false);

function initAdvert() {
     var adBox = "annoyingAdvert";

     document.getElementById(adBox).style.display = "block";
     document.getElementById(adBox).addEventListener("mouseover",slide,false);
     document.getElementById("closeBox").addEventListener(
       "click",
       function() {
          document.getElementById(adBox).style.display = "none";
     },
     false
   );
}

function slide() {
     var adBox = "annoyingAdvert";

     if (nextPos(adBox) <= (document.body.clientWidth-150)) {
        document.getElementById(adBox).style.left = nextPos(adBox) + "px";
        setTimeout(slide,100);
     }

     function nextPos(elem) {
        return document.getElementById(elem).offsetLeft+1;
     }
}
```

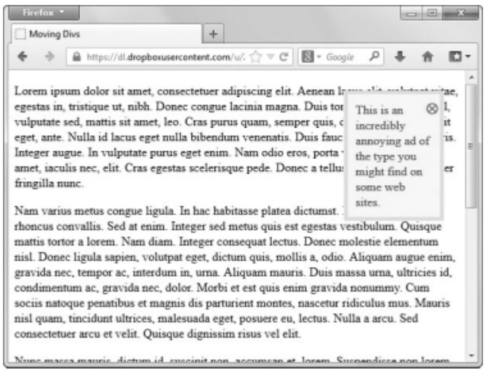


图 11-8　在这个版本中，广告会移动右边，可以在这里最终关闭它

⇛ 移动对象

1. `document.getElementById(adBox).addEventListener("mouseover",slide,false);`

为了开始移动，在广告上添加一个 mouseover 事件处理程序，让它触发 slide()函数。

2. `if (nextPos(adBox) <= (document.body.clientWidth-150)) {`

在移动广告层之前，需要检查它的位置是否处于限制范围内——实现的方法是获取它的当前位置（使用 nextPos()函数）并与文档窗口的宽度作比较。如果当前位置小于窗口宽度（减去 150 像素是为了把层本身的宽度考虑进去），那么就希望继续移动。

3. `document.getElementById(adBox).style.left = nextPos(adBox) + "px";`

为了以适合各种浏览器的方式移动广告层，我们必须修改它的 style.left 属性。在这里，获取对象的下一个位置，最后添加 px，这样就形成了 style.left 属性所需的格式。只需修改 style.left 属性，对象就会移动到新的位置。

4. `setTimeout(slide,100);`

这里，通过调用 setTimeout()，每 100 毫秒（0.1 秒）重复调用 slide()，从而让 JavaScript 一直移动。

5. `function nextPos(elem) {`

前面有两个地方用到了元素的当前位置，这里的函数用来获得当前位置。这个函数需要的参数只是元素的 id。

6. `return document.getElementById(elem).offsetLeft+1;`

得到了对象的 id，就可以找到这个对象。然后，只需获取对象的 offsetLeft 属性，这是对象的左边位置。offsetLeft 属性包含一个数字值，所以可以按原样返回它。

✔提示

❑ 你可能会奇怪：既然 offsetLeft 是数字，为什么不直接设置它，而要修改 style.left 属性？必须这么做，因为 offsetLeft 是只读的，也就是说，可以读它的值，但是不能修改它。没有任何跨浏览器的可写的数字型定位元素。

11

11.9 日期方法

因为常常需要处理日期，所以这里用一个表格列出 Date 对象的所有方法。在表 11-1 中，UTC 是 Coordinated Universal Time 的缩写，它在 1986 年取代格林尼治时间，成为时间的世界标准。包含 UTC 的方法只能在 JavaScript 1.2 或更高版本中使用。

表11-1 日期方法

方　法	描　述	返 回 值	JavaScript版本
getDate() getUTCDate()	月中的日期	1~31	1.0 1.2
getday() getUTCDay()	周中的日期（整数值）	0~6	1.0 1.2
getFullYear() getUTCFullYear()	完整的四位年份	1900+	1.2
getHours() getUTCHours()	日中的时期（整数值）	0~23	1.0 1.2

（续）

方 法	描 述	返 回 值	JavaScript版本
getMilliseconds() getUTCMilliseconds()	自上一秒以来的毫秒数	0~999	1.2
getMinutes() getUTCMinutes()	自上一小时以来的分钟数	0~59	1.0 1.2
getMonth() getUTCMonth()	年中的月份	0~11	1.0 1.2
getSeconds() getUTCSeconds()	自上一分钟以来的秒数	0~59	1.0 1.2
getTime()	自1970年1月1日午夜以来的毫秒数		1.0
getTimezoneOffset()	本地时间和GMT之间相差的分钟数	0~1439	1.0
getYear()	日期的年份部分	对于1900~1999年的年份，是0~99；对于以后的年份，是四位数字	1.0
parse()	给出一个日期/时间字符串，返回自1970年1月1日午夜以来的毫秒数		1.0
setDate() setUTCDate()	给出1~31之间的数字，设置天	以毫秒数表示的日期（从Java Script 1.2开始）	1.0 1.2
setFullYear() setUTCFullYear()	给出四位的年份，设置年份	以毫秒数表示的日期	1.2
setHours() setUTCHours()	给出0~23之间的数字，设置小时	以毫秒数表示的日期（从Java Script 1.2开始）	1.0 1.2
setMilliseconds() setUTCMilliseconds()	给出0~999之间的数字，设置毫秒	以毫秒数表示的日期	1.0 1.2
setMinutes() setUTCMinutes()	给出0~59之间的数字，设置分钟	以毫秒数表示的日期（从Java Script 1.2开始）	1.0 1.2
setMonth() setUTCMoth()	给出0~11之间的数字，设置月份	以毫秒数表示的日期（从Java Script 1.2开始）	1.0 1.2
setSeconds() setUTCSeconds()	给出0~59之间的数字，设置秒	以毫秒数表示的日期（从Java Script 1.2开始）	1.0 1.2
setTime()	给出自1970年1月1日午夜以来的毫秒数，设置日期	以毫秒数表示的日期	1.0
setYear()	给出两位或四位数字值，设置年份	以毫秒数表示的日期（从Java Script 1.2开始）	1.0
toGMTString() toUTCString()	字符串格式的GMT日期和时间	day dd mm yyyy hh:mm:ss GMT	1.0 1.2
toLocaleString()	字符串格式的本地日期和时间	根据操作系统、地区和浏览器的不同，返回值有所不同	1.0
toString()	字符串格式的本地日期和时间	根据操作系统和浏览器的不同，返回值有所不同	1.0
UTC()	给出年、月、日（以及可选的小时、分钟、秒和毫秒）格式的日期，返回自1970年1月1日以来的毫秒数	以毫秒数表示的日期	1.0
valueOf()	自1970年1月1日午夜以来的毫秒数	以毫秒数表示的日期	1.2

第 12 章 JavaScript 应用示例

12

在本书前面的各章中，你学习了如何使用许多 JavaScript 技术实现特定的任务。在你建立的许多网页上，常常只需要使用一种技术，而且能够使用本书中的脚本来完成任务（常常只需要作少量修改）。

但是，有时候需要使用多种技术才能完成任务，这就是本章涉及的主题。本章中的任务需要结合使用各种技术，而且这些任务在你自己的 Web 站点上往往很常见。

在本章中，你将学习如何用大纲式可折叠菜单和下拉菜单改进站点的用户界面，创建幻灯片，用 JavaScript 处理文本，让 JavaScript 以容易理解的图形化方式显示数据，以及在脚本的控制下在不同的样式表之间切换。

12.1 使用可折叠菜单

可折叠菜单（sliding menu）是一种简单的用户界面组件，可以利用它在页面上放置许多信息，同时又不会使页面显得太混乱。用户每次可以只在需要时查看他们想要看的额外信息。脚本 12-1 包含 HTML，脚本 12-2 包含 CSS，脚本 12-3 包含 JavaScript，如下所示。

脚本 12-1 这是一个包含许多链接的简单 HTML 页面

```
<!DOCTYPE html>
<html>
<head>
     <title>Shakespeare's Plays</title>
     <link rel="stylesheet" href="script01.css">
     <script src="script01.js"></script>
</head>
<body>
     <h1>Shakespeare's Plays</h1>
     <div>
        <a href="menu1.html" class="menuLink">Comedies</a>
        <ul class="menu" id="menu1">
           <li><a href="pg1.html">All's Well That Ends Well</a></li>
           <li><a href="pg2.html">As You Like It</a></li>
           <li><a href="pg3.html">Love's Labour's Lost</a></li>
           <li><a href="pg4.html">The Comedy of Errors</a></li>
        </ul>
     </div>
     <div>
```

```
            <a href="menu2.html" class="menuLink">Tragedies</a>
            <ul class="menu" id="menu2">
               <li><a href="pg5.html">Anthony &amp; Cleopatra</a></li>
               <li><a href="pg6.html">Hamlet</a></li>
               <li><a href="pg7.html">Romeo &amp; Juliet</a></li>
            </ul>
         </div>
         <div>
            <a href="menu3.html" class="menuLink">Histories</a>
            <ul class="menu" id="menu3">
               <li><a href="pg8.html">Henry IV, Part 1</a></li>
               <li><a href="pg9.html">Henry IV, Part 2</a></li>
            </ul>
         </div>
    </body>
    </html>
```

脚本 12-2　实现可折叠菜单效果并不需要很多 CSS

```
body {
     background-color: #FFF;
     color: #000;
}

div {
     margin-bottom: 10px;
}

ul.menu {
     display: none;
     list-style-type: none;
     margin-top: 5px;
}

a.menuLink {
     font-size: 1.2em;
     font-weight: bold;
}
```

脚本 12-3　这个脚本可以控制文本（和链接）的出现和消失

```
window.addEventListener("load",initAll,false);

function initAll() {
     var allLinks = document.getElementsByTagName("a");

     for (var i=0; i<allLinks.length; i++) {
        if (allLinks[i].className.indexOf("menuLink") > -1) {
           allLinks[i].addEventListener("click",toggleMenu,flase);
        }
     }
}

function toggleMenu(evt) {
     var startMenu = this.href.lastIndexOf("/")+1;
     var stopMenu = this.href.lastIndexOf(".");
     var thisMenuName = this.href.substring(startMenu,stopMenu);

     var thisMenuStyle = document.getElementById(thisMenuName).style;
     if (thisMenuStyle.display == "block") {
```

```
            thisMenuStyle.display = "none";
        }
        else {
            thisMenuStyle.display = "block";
        }

        evt.preventDefault();
  }
```

⇛ 使用可折叠菜单

1. `var allLinks = document.getElementsByTagName("a");`

当页面加载时，调用 `initAll()`函数，该函数首先创建一个包含页面上所有链接的数组。

2.
```
for (var i=0; i<allLinks.length; i++) {
    if (allLinks[i].className.indexOf("menuLink") > -1) {
      allLinks[i].addEventListener("click",toggleMenu,false);
    }
  }
```

在找到了所有链接之后，对它们进行遍历，查找具有 `menuLink` 类的链接，并且给这些链接添加 `click` 处理程序。在这里，`click` 处理程序设置为在单击链接时调用 `toggleMenu()`函数。

3.
```
var startMenu = this.href.lastIndexOf("/")+1;
  var stopMenu = this.href.lastIndexOf(".");
```

在 `toggleMenu()`中，JavaScript 给出了 `this`。在这里，`this` 是用户单击的链接对象，因此 `this.href` 是完整的链接 URL。但是，我们只需要最后一个正斜杠和最后一个点号之间的部分（也就是说，如果链接指向 http://www.javascriptworld.com/index.html，那么我们只需要 index），所以我们创建 `startMenu` 和 `stopMenu` 变量，并且将它们设置为 `this.href` 中的两个位置，我们将根据这两个位置找到组成菜单名的字符串。

4. `var thisMenuName = this.href.substring(startMenu,stopMenu);`

我们需要的菜单名在这两个位置之间，所以在这里设置它。

5. `var thisMenuStyle = document.getElementById(thisMenuName).style;`

我们使用 `getElementById()`方法将 `thisMenuStyle` 变量设置为所需菜单的 `style` 属性。

6.
```
if (thisMenuStyle.display == "block") {
    thisMenuStyle.display = "none";
  }
  else {
    thisMenuStyle.display = "block";
  }
```

如果 `thisMenuStyle` 的 `display` 属性是 `block`，那么这行代码将它设置为 `none`。或者如果它是 `none`，就把它改为 `block`。这样就会在菜单的显示状态和折叠状态之间进行切换，见图 12-1 和图 12-2。



图 12-1　可折叠菜单的初始视图



图 12-2　在单击之后，菜单展开，额外的选择出现了

7. `evt.preventDefault();`

这个函数的最后，是阻止对已点击链接的任何进一步处理，用上面这条语句即可实现。

✓提示

- 菜单背后的原理如下：如果仔细查看 HTML，你会发现链接其实是在`<ul>`和`<li>`标签中，也就是说，它们是无序列表和列表项。如果用户的浏览器不支持 CSS，那么他们只会在网页上看到如图 12-3 所示的列表。而如果浏览器支持 CSS，我们就可以使用 CSS 来为链接设置样式，使其看起来不再是普通列表。



图 12-3　如果关闭 CSS 显示，那么菜单就会显示出其本来面目——简单的无序列表

12.2　添加下拉菜单

看一下前一节中的可折叠菜单，你可能会认为："这很不错，但我想要的其实是下拉菜单，这会让网页看起来像应用程序。"其实，在前一个示例和下拉菜单之间并没有很大差异。实际上，HTML 是相同的（请参考脚本 12-1），CSS 见脚本 12-4，JavaScript 见脚本 12-5。

脚本 12-4　只需多加一点儿 CSS 和 JavaScript，就可以让菜单具有更传统的外观

```
body {
     background-color: #FFF;
     color: #000;
}

div {
     margin-bottom: 10px;
     width: 15em;
     background-color: #9CF;
}

ul.menu {
     display: none;
     list-style-type: none;
     margin: 0;
     padding: 0;
}

ul.menu li {
     font: 1em arial, helvetica, sans-serif;
     padding-left: 10px;
}

a.menuLink, li a {
     text-decoration: none;
     color: #006;
}

a.menuLink {
     font-size: 1.2em;
     font-weight: bold;
}

ul.menu li a:hover {
     background-color: #006;
     color: #FFF;
     padding-right: 10px;
}
```


脚本 12-5　这个脚本将一般的链接转换为下拉菜单

```
window.addEventListener("load",initAll,false);

function initAll() {
     var allLinks = document.getElementsByTagName("a");

     for (var i=0; i<allLinks.length; i++) {
        if (allLinks[i].className.indexOf("menuLink") > -1) {
           allLinks[i].addEventListener("mouseover",toggleMenu,false);
           allLinks[i].addEventListener("click",clickHandler,false);
        }
     }
}

function clickHandler(evt) {
     evt.preventDefault();
}
```

```
function toggleMenu() {
     var startMenu = this.href.lastIndexOf("/")+1;
     var stopMenu = this.href.lastIndexOf(".");
     var thisMenuName = this.href.substring(startMenu,stopMenu);

     var menuParent = document.getElementById(thisMenuName).parentNode;
     var thisMenuStyle = document.getElementById(thisMenuName).style;
     thisMenuStyle.display = "block";

     menuParent.addEventListener("mouseout",function() {thisMenuStyle.display = "none";},false);
     menuParent.addEventListener("mouseover",function() {thisMenuStyle.display = "block";},false);
}
```

⇒ 添加下拉菜单

1. `allLinks[i].addEventListener("mouseover",toggleMenu,false);`
   `allLinks[i].addEventListener("click",clickHandler,false);`

在这里，并不像前面那样添加 click 处理程序来调用 toggleMenu()，而是让 click 调用 clickHandler()。我们将让 onmouseover 调用 toggleMenu()，这意味着当用户将鼠标移动到这个链接上时，菜单就会展开（见图 12-4）。



图 12-4　当把鼠标移动到链接上时，菜单就会展开，并且突出显示鼠标指针下的选项

2.
```
function clickHandler(evt) {
   evt.preventDefault();
}
```

clickHandler()函数只需要确保不会对被单击的链接进行进一步的处理。

3.
```
var menuParent = document.getElementById(thisMenuName).parentNode;
var thisMenuStyle = document.getElementById(thisMenuName).style;
thisMenuStyle.display = "block";
```

让菜单显示之后，必须知道什么时候应该让它消失。下拉菜单的特点是，当鼠标离开触发菜单的链接时，并不希望关闭菜单；只有在鼠标离开整个 div 时，才应该关闭菜单。也就是说，如果鼠标还在菜单上的任何位置，它就应该打开着。为此，我们需要菜单名称节点的父节点，将其存储在 menuParent 变量中。

最后，我们不再想以同样的方式切换菜单，而要将菜单设置为一直显示。

4. `menuParent.addEventListener("mouseout",function() {thisMenuStyle.display = "none";}, false);`

有一个技巧：如果执行到这里，那么根据定义，鼠标指针应该已经在 `div` 中了。因此，只需设置父 `div` 的 `mouseover` 事件处理程序，就会马上触发它。

5. `menuParent.addEventListener("mouseover",function() {thisMenuStyle.display = "block";},`
`➝false);`

这里让整个 `div`（再次）显示。的确，我们在步骤 3 中已经这样做了，但是我们需要在这里再做一次，否则当鼠标离开链接时，菜单就会关闭。

12.3 改进下拉菜单

看到前一个示例之后，你可能会说，“我不想要垂直的菜单，我要一个水平菜单！”这非常简单（甚至不需要修改 JavaScript）！你还可能希望给习惯使用键盘进行导航的用户提供一些方便。下面就看看如何实现这些需求。同样，不需要修改 HTML，如果想看 HTML，翻回去看脚本 12-1 就可以了。

脚本 12-6 只需在 CSS 中添加一行，菜单的外观就完全不一样了

```
body {
     background-color: #FFF;
     color: #000;
}

div {
     margin-bottom: 10px;
     width: 15em;
     background-color: #9CF;
     float: left;
}

ul.menu {
     display: none;
     list-style-type: none;
     margin: 0;
     padding: 0;
}

ul.menu li {
     font: 1em arial, helvetica, sans-serif;
     padding-left: 10px;
}

a.menuLink, li a {
     text-decoration: none;
     color: #006;
}

a.menuLink {
     font-size: 1.2em;
     font-weight: bold;
}

ul.menu li a:hover {
     background-color: #006;
     color: #FFF;
```

12

```
    padding-right: 10px;
}
```

脚本 12-7 添加一点儿 JavaScript 代码，让用户不用鼠标也能够操作这个菜单

```
window.addEventListener("load",initAll,false);

function initAll() {
     var allLinks = document.getElementsByTagName("a");

     for (var i=0; i<allLinks.length; i++) {
        if (allLinks[i].className.indexOf("menuLink") > -1) {
           allLinks[i].addEventListener("mouseover",toggleMenu,false);
           allLinks[i].addEventListener("click",clickHandler,false);
        }
     }
}

function clickHandler(evt) {
     if (evt) {
        evt.preventDefault();

        if (typeof evt.target == "string") {
           toggleMenu(evt,evt.target);
        }
        else {
           toggleMenu(evt,evt.target.toString());
        }
     }
     else {
        toggleMenu(evt,window.event.srcElement.href);
     }
}

function toggleMenu(evt,currMenu) {
     if (toggleMenu.arguments.length < 2) {
        var currMenu = this.href;
     }

     var startMenu = currMenu.lastIndexOf("/")+1;
     var stopMenu = currMenu.lastIndexOf(".");
     var thisMenuName = currMenu.substring(startMenu,stopMenu);

     var menuParent = document.getElementById(thisMenuName).parentNode;
     var thisMenuStyle = document.getElementById(thisMenuName).style;
     thisMenuStyle.display = "block";

     menuParent.addEventListener("mouseout",function() {thisMenuStyle.display = "none";},false);
     menuParent.addEventListener("mouseover",function() {thisMenuStyle.display = "block";},false);
}
```



图 12-5 对 CSS 的一个简单修改就使菜单横跨页面，而不是在左边垂直排列

⇛ 改进下拉菜单

1. `float: left;`

很简单，只需在每个菜单 `div` 的 CSS 中添加 `float:left;`，见脚本 12-6 和图 12-5。这样就可以把垂直菜单转换为水平菜单，不需要修改 JavaScript。

2. `allLinks[i].addEventListener("click",clickHandler,false);`

但是，如果希望使菜单的可操作性更强，就需要添加一点儿 JavaScript 代码，见脚本 12-7。具体地说，需要在 `click` 事件处理程序中添加代码，所以把它设置为函数 `clickHandler`。

3. `function clickHandler(evt) {`

这是事件处理程序，传递的是 `evt` 参数。正如前几个示例中提到的，有些浏览器传递事件对象，有些不传递。

4.
```
if (evt) {
   evt.preventDefault();

   if (typeof evt.target == "string") {
     toggleMenu(evt,evt.target);
   }
   else {
     toggleMenu(evt,evt.target.toString());
   }
}
else {
   toggleMenu(evt,window.event.srcElement.href);
}
```

这些代码处理浏览器在事件传递机制方面的差异。首先，检查是否有事件对象，也就是 `evt` 是否存在。如果有事件对象存在，就检查它的 `target` 属性是否是一个字符串，因为我们需要字符串形式的目标。如果它是字符串，就把事件和它的目标传递给 `toggleMenu()`。

如果 `target` 属性不是字符串，就通过调用 `toString()`方法把它转换为字符串，然后使用这个字符串（和 `evt`）作为 `toggleMenu()`的参数。

最后，如果没有事件对象，就向 `toggleMenu()`传递一个伪 `evt` 对象和 `window.event.srcElement.href`（这是 IE 存储我们需要的值的位置）。

5.
```
function toggleMenu(evt,currMenu) {
   if (toggleMenu.arguments.length < 2) {
     var currMenu = this.href;
   }
```

这段代码负责处理菜单的显示和隐藏，因为单击和鼠标移动都可以触发菜单的显示，所以 toggleMenu()比较复杂。这个函数有两个参数，但是 JavaScript 的函数有一个重要的特点：即使函数期望传递两个参数，也并不意味着必须传递两个参数。实际上，由于我们编写 toggleMenu()的方式，它可以应对 3 种情况。

- 当浏览器是 IE 并通过鼠标触发 toggleMenu()时，不传递参数。
- 当浏览器不是 IE 并通过鼠标触发 toggleMenu()时，传递一个参数（event 对象）。
- 当通过 clickHandler()调用 toggleMenu()时，传递两个参数（event 对象和菜单名）。

如果不传递参数或只传递一个参数（可以通过查看 toggleMenu.arguments.length 来检查），就说明可以通过 this.href 找到菜单名，换句话说，应该采用与以前一样的方式。但是，因为需要这个值在 currMenu 中，所以要存储它。

6.
```
var startMenu = currMenu.lastIndexOf("/")+1;
var stopMenu = currMenu.lastIndexOf(".");
var thisMenuName = currMenu.substring(startMenu,stopMenu);
```

同样，要计算 startMenu、stopMenu 和 thisMenuName，但是这一次是根据 currMenu 进行计算。

✔提示

- 需要使用键盘访问菜单项的不仅仅是视障人士。有些人喜欢使用键盘，而且有些浏览器（比如移动设备中的浏览器）无法按照菜单需要的方式处理鼠标移动。考虑可访问性总是好想法，使用 JavaScript 或新奇的特性不应该成为忽视各种用户的特殊需要的借口。
- 在这个示例中，在菜单项上敲击键盘会展开菜单，但是关闭菜单需要使用鼠标。

12.4　带说明的幻灯片

尽管像脚本 4-18 和脚本 4-19 那样的幻灯片很方便，但是如果可以显示随图像改变的说明，就更好了。脚本 12-8（HTML）、脚本 12-9（CSS）和脚本 12-10（JavaScript）给出一个这样的幻灯片示例（显示的是我们夏天度假时拍的照片）。在这个示例中，我们将演示如何将前面见过的不同技术组合在一个脚本中。

脚本 12-8　幻灯片 HTML 页面

```
<!DOCTYPE html>
<html>
<head>
     <title>Our Summer Vacation!</title>
     <link rel="stylesheet" href="script04.css">
     <script src="script04.js"></script>
</head>
<body>
     <h1>Our Summer Vacation Slideshow</h1>
     <img src="images/slideImg0.jpg" alt="Our Vacation Pix" id="slideshow">
     <div id="imgText"> </div>
     <form id="chgImg">
        <input type="button" id="prevLink" value="&laquo; Previous">
        <input type="button" id="nextLink" value="Next &raquo;">
     </form>
</body>
</html>
```

脚本 12-9　脚本 12-8 调用的外部 CSS

```
body {
     background-color: #FFF;
     color: #000;
     font: 12px verdana, arial, helvetica, geneva, sans-serif;
}

h1 {
     font: 24px "trebuchet ms", verdana, arial, helvetica, geneva, sans-serif;
     margin-left: 100px;
}

#chgImg {
     margin-left: 100px;
     clear: both;
}

#slideshow {
     padding: 0 10px 10px 10px;
     float: left;
     height: 240px;
     width: 320px;
}

#imgText {
     padding: 10px 0 0 10px;
     float: left;
     width: 200px;
     height: 150px;
     border-top: 1px solid #000;
     border-left: 1px solid #000;
}
```

脚本 12-10　这个幻灯片脚本显示照片和说明

```
window.addEventListener("load",initAll,false);

var currImg = 0;
var captionText = [
     "Our ship, leaving Vancouver.",
     "We took a helicopter ride at our first port, Juneau.",
     "The helicopter took us to Mendenhall Glacier.",
     "The happy (and chilly) couple, on the glacier.",
     "Here's what our second stop, Ketchikan, looked like from the ship.",
     "We got to cruise through Glacier Bay. It was absolutely breathtaking!",
     "In Skagway, we took a train up into the mountains, all the way to the Canadian Border.",
     "Looking back down at Skagway from the train.",
     "On a trip this romantic, I shouldn't have been surprised by a proposal, but I was (obviously,
     ➝I said yes).",
     "It's nice to go on vacation, but it's nice to be home again, too."
];

function initAll() {
     document.getElementById("imgText").innerHTML = captionText[currImg];
     document.getElementById("prevLink").addEventListener("click",function() {newSlide(-1);},false);
     document.getElementById("nextLink").addEventListener("click",function() {newSlide(1);},false);
}

function newSlide(direction) {
     var imgCt = captionText.length;
```

```
        currImg = currImg + direction;
        if (currImg < 0) {
           currImg = imgCt-1;
        }
        if (currImg == imgCt) {
           currImg = 0;
        }
        document.getElementById("slideshow").src = "images/slideImg" + currImg + ".jpg";
        document.getElementById("imgText").innerHTML = captionText[currImg];
}
```

⇒ 创建带说明的幻灯片

1. `document.getElementById("imgText").innerHTML = captionText[currImg];`

initAll()函数需要设置 3 个东西：第一张幻灯片的照片说明（放在 imgText 区域中），以及前进和后退按钮的 click 处理程序（见下面的步骤）。

2. `document.getElementById("prevLink").addEventListener("click",function(){newSlide(-1);},`
   `→false);`
   `document.getElementById("nextLink").addEventListener("click",function(){newSlide(1);},`
   `→false);`

这就是这两个函数的全部（至少是大部分）操作。我们可以建立复杂的代码，根据单击的按钮决定是前进还是后退。但是不需要这么做，只需建立两个都调用 newSlide()的函数。这两个函数调用的区别是：一个调用传递 1，另一个调用传递-1，这个参数让 newSlide()知道移动的方向。

3. `document.getElementById("slideshow").src = "images/slideImg" + currImg + ".jpg";`
   `document.getElementById("imgText").innerHTML = captionText[currImg];`

这个步骤同时改变图像和对应的说明，图 12-6 显示其结果。

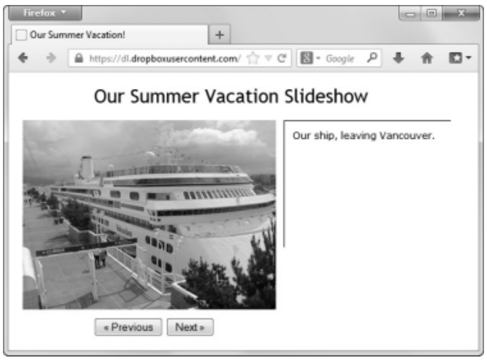


图 12-6　这个脚本决定要显示的照片和说明

✔提示

- 对 captionText 及其语法感到疑惑吗？回忆一下之前我们说过的：条条大路通罗马，在 JavaScript 中，声明新数组的方法不止一种，这一点儿都不奇怪。下面两种声明方式的结果是一模一样的：

```
var dice = new Array(1,2,3,4,5,6);
```

和

```
var dice = [1,2,3,4,5,6];
```

两种方法都可以生成一个包含六面骰子可选值的数组。

12.5　一个娱乐姓名生成器

你以前可能见过一些娱乐性的 Web 程序，它们接受你的姓名并且将它转换为一个新姓名，比如超级英雄的姓名或《黑道家族》(*The Sopranos*) 影剧角色的姓名。我们想证明这是很荒谬的，所以通过脚本 12-11 和脚本 12-12 演示如何生成这种傻乎乎的姓名。在这个过程中，你会看到如何将字符串处理、数组、错误检查和表单验证组合成一个无聊的脚本。

脚本 12-11　可以在这个网页中输入你的真实姓名，然后就会得到娱乐姓名

```
<!DOCTYPE html>
<html>
<head>
     <title>Silly Name Generator</title>
     <script src="script05.js"></script>
</head>
<body>
<h1>What's your silly name?</h1>
<table>
     <tr>
        <td class="rtAlign">First Name:</td>
        <td><input type="text" id="fName" size="30"></td>
     </tr>
     <tr>
        <td class="rtAlign">Last Name:</td>
        <td><input type="text" id="lName" size="30"></td>
     </tr>
     <tr>
        <td> </td>
        <td><input type="submit" value="Submit" id="sillySubmit"></td>
     </tr>
</table>
<p id="msgField"> </p>
</body>
</html>
```

脚本 12-12　这个脚本根据用户输入的名和姓中的字符，从三个数组生成娱乐姓名

```
window.addEventListener("load",initAll,false);

function initAll() {
     document.getElementById("sillySubmit").addEventListener(
        "click",
        function() {
           document.getElementById("msgField").innerHTML = getSillyName();
           return false;
        },
        false
     );
}
```

```
function getSillyName() {
    var firstName = ["Runny", "Buttercup","Dinky", "Stinky", "Crusty", "Greasy","Gidget", "Cheesypoof",
    ➝"Lumpy","Wacky", "Tiny", "Flunky", "Fluffy","Zippy", "Doofus", "Gobsmacked","Slimy", "Grimy",
    ➝"Salamander","Oily", "Burrito", "Bumpy", "Loopy","Snotty", "Irving", "Egbert"];
    var lastName1 = ["Snicker", "Buffalo","Gross", "Bubble", "Sheep", "Corset","Toilet", "Lizard",
    ➝"Waffle","Kumquat", "Burger", "Chimp", "Liver","Gorilla", "Rhino", "Emu", "Pizza","Toad",
    ➝"Gerbil", "Pickle", "Tofu","Chicken", "Potato", "Hamster","Lemur", "Vermin"];
    var lastName2 = ["face", "dip", "nose","brain", "head", "breath", "pants","shorts", "lips",
    ➝"mouth", "muffin","butt", "bottom", "elbow", "honker","toes", "buns", "spew", "kisser","fanny",
    ➝"squirt", "chunks", "brains","wit", "juice", "shower"];

    var firstNm = document.getElementById("fName").value.toUpperCase();
    var lastNm = document.getElementById("lName").value.toUpperCase();
    var validName = true;

    if (firstNm == "") {
       validName = false;
    }
    else {
       var firstNum = firstNm.charCodeAt(0) - 65;
       if (firstNum < 0 || firstNum > 25) {
          validName = false;
       }
    }

    if (!validName) {
       document.getElementById("fName").focus();
       document.getElementById("fName").select();
       return "That's not a valid first name";
    }

    if (lastNm == "") {
       validName = false;
    }
    else {
       var lastNum1 = lastNm.charCodeAt(0) - 65;
       var lastNum2 = lastNm.charCodeAt(lastNm.length-1) - 65;

       if (lastNum1 < 0 || lastNum1 > 25 || lastNum2 < 0 || lastNum2 > 25) {
          validName = false;
       }
    }

    if (!validName) {
       document.getElementById("lName").focus();
       document.getElementById("lName").select();
       return "That's not a valid last name";
    }

    return "Your silly name is " + firstName [firstNum] + " " + lastName1[lastNum1] + lastName2
       [lastNum2];
}
```

⇛ 组合 JavaScript 技术

```
1. document.getElementById("msgField").innerHTML = getSillyName();
   return false;
```

当最初加载页面时，将 submit 按钮的 `click` 处理程序设置为调用一个函数，这里是这个函数的内容。首先，调用 `getSillyName()`。这个函数返回一个字符串值（要么是娱乐名，要么是一个错误消息），

我们会把这个字符串写到页面上。然后返回 false，这样 click 就不会尝试将表单提交到服务器。

2.
```
var firstNm = document.getElementById("fName").value.toUpperCase();
var lastNm = document.getElementById("lName").value.toUpperCase();
```

页面要求每个访问者在文本字段中输入他们的名字和姓氏。当提交表单时，在 getSillyName()函数中，首先将名字和姓氏转换为全大写，并且将结果存储在变量 firstNm 和 lastNm 中。

3.
```
if (firstNm = ="") {
  validName = false;
}
```

页面要求访问者第一次在名字字段中至少输入一个字符，所以在这里进行检查。请记住，这个表达式的意义是“如果 firstNm 为空，那么……”。如果是这种情况，就将 validName 设置为 false。

4.
```
var firstNum = firstNm.charCodeAt(0) - 65;
```

否则，charCodeAt()方法从字符串中取出一个字符。这个字符的位置取决于传递给方法的数字参数。在这个示例中，取出 0 位置上的字符，也就是字符串中的第一个字符（请记住，JavaScript 从 0 开始编号），并且返回这个字符的 ASCII 值。大写字母 A 的 ASCII 值为 65，Z 的 ASCII 值为 90。所以将返回值减 65 应该获得 0 ~ 25 的值，将结果保存在 firstNum 中。

5.
```
if (firstNum < 0 || firstNum > 25) {
  validName = false;
}
```

如果用户输入的名字并非以 A ~ Z 的字符开头，那么就没有对应的娱乐名。所以，在检查姓氏之前，要确保首字母处于这个范围中。如果不满足这个条件，就将 validName 设置为 false。

6.
```
if (!validName) {
  document.getElementById("fName").focus();
  document.getElementById("fName").select();
  return "That's not a valid first name";
}
```

在这里检查 validName，如果 validName 是 false，就意味着用户输入的名字是无效的。当发生这种情况时，我们将输入光标放进这个字段，选择这个字段中的所有内容，并且返回上一个错误消息。

7.
```
if (lastNm == "") {
  validName = false;
}
```

与名字一样，访问者必须在姓氏字段中输入一些内容。

8.
```
var lastNum1 = lastNm.charCodeAt(0) - 65;
var lastNum2 = lastNm.charCodeAt(lastNm.length-1) - 65;
```

为了找到访问者的娱乐姓氏，需要找到姓氏的第一个字符和最后一个字符的 ASCII 值。第一个字符的处理与步骤 4 中相似。最后一个字符是通过 lastNm 字符串的长度减 1，然后将这个数字传递给 charCodeAt()方法找到的。

12

9.
```
if (lastNum1 < 0 || lastNum1 > 25 || lastNum2 < 0 || lastNum2 > 25) {
    validName = false;
}
```

与名字字段一样，必须确保姓氏的第一个字符和最后一个字符包含 A ~ Z 的字符；如果不满足条件，就将 validName 再次设置为 false。

10.
```
if (!validName) {
    document.getElementById("lName").focus();
    document.getElementById("lName").select();
    return "That's not a valid last name";
}
```

与第 6 步中一样，如果姓氏是无效的，就向用户显示错误消息。

11.
```
return "Your silly name is " + firstName[firstNum] + " " + lastName1[lastNum1] +
➝lastName2[lastNum2];
```

如果通过了所有测试，就要计算这个新的娱乐名了。因为已经将字符转换为 0 ~ 25 的数字，所以可以将结果用作 firstName、lastName1 和 lastName2 数组的索引。将每个数组查找的结果依次拼接起来，并且在名字和姓氏之间加一个空格。注意，姓氏的两部分是直接拼接的，没有加空格。完成之后，将这个姓名返回并放进文档中，见图 12-7。



图 12-7　产生的娱乐名

娱乐名

寻找娱乐名的方法是，获得名字的第一个字母、姓氏的第一个字母和姓氏的最后一个字母，然后在表 12-1 中查找它们。通过名字的第一个字母找到新的名字，通过姓氏的头尾字母找到新姓氏的两个部分。

例如，Tom 中的 T 对应于新名字 Oily，Negrino 中的 N 和 O 产生新姓氏 Gorillahonker。Dori 中的 D 转换为 Stinky，Smith 中的 S 和 H 转换为 Gerbilshorts。因此，本书作者的娱乐名是 Oily Gorillahonker 和 Stinky Gerbilshorts。

表12-1 娱乐名表

	名字的第一个字母	姓氏的第一个字母	姓氏的最后一个字母
A	Runny	Snicker	face
B	Buttercup	Buffalo	dip
C	Dinky	Gross	nose
D	Stinky	Bubble	brain
E	Crusty	Sheep	head
F	Greasy	Corset	breath
G	Gidget	Toilet	pants
H	Cheesypoof	Lizard	shorts
I	Lumpy	Waffle	lips
J	Wacky	Kumquat	mouth
K	Tiny	Burger	muffin
L	Flunky	Chimp	butt
M	Fluffy	Liver	bottom
N	Zippy	Gorilla	elbow
O	Doofus	Rhino	honker
P	Gobsmacked	Emu	toes
Q	Slimy	Pizza	buns
R	Grimy	Toad	spew
S	Salamander	Gerbil	kisser
T	Oily	Pickle	fanny
U	Burrito	Tofu	squirt
V	Bumpy	Chicken	chunks
W	Loopy	Potato	brains
X	Snotty	Hamster	wit
Y	Irving	Lemur	juice
Z	Egbert	Vermin	shower

12.6 柱状图生成器

图表是显示可视信息的好方法。可以在 Adobe Photoshop 中绘制柱状图，或者在 Microsoft Excel 中计算它们，但是对于可能需要定期更新的动态数据，为什么不用 JavaScript 实时地生成它们呢？我们在本书中一直在说 JavaScript 是面向对象的，而且在本书中许多地方都使用了对象，但是只简要地解释了如何创建自己的自定义对象。下面这个示例更深入地演示对象的工作方式，见脚本 12-13（HTML）、脚本 12-14（CSS）和脚本 12-15（JavaScript）。

脚本 12-13　柱状图生成器的 HTML 页面

```
<!DOCTYPE html>
<html>
<head>
     <title>Bar Chart Display</title>
     <link rel="stylesheet" href="script06.css">
     <script src="script06.js"></script>
</head>
<body>
<div id="chartType">
     Choose a chart:<br>
     <input type="radio" name="type" value="browser" checked>Browser Usage<br>
     <input type="radio" name="type" value="platform"> Mobile Device Vendors<br>
     <p><br></p>
     Choose a color:<br>
     <input type="radio" name="color" value="lilRed.gif" checked> Red<br>
     <input type="radio" name="color" value="lilGreen.gif"> Green<br>
     <input type="radio" name="color" value="lilBlue.gif"> Blue<br>
     <p><br></p>
     Choose a direction:<br>
     <input type="radio" name="direction" value="horizontal"  checked> Horizontal<br>
     <input type="radio" name="direction" value="vertical"> Vertical
</div>
<div id="chartArea"> </div>
</body>
</html>
```

脚本 12-14　这个脚本包含柱状图示例的样式

```
body {
     background-color: #FFF;
     color: #000;
     font-size: 12px;
}

#chartType {
     float: left;
     width: 200px;
}

th {
     font-size: 16px;
     padding-left: 20px;
     padding-right: 15px;
}

.horiz th {
     border-right: 1px #000 solid;
}

.horiz img {
     height: 15px;
     vertical-align: bottom;
     margin-right: 10px;
}

.vert {
     text-align: center;
     vertical-align: bottom;
```

```
}

.vert th {
     border-left: 1px #000 solid;
     border-bottom: 1px #000 solid;
}

.vert img {
     width: 15px;
     padding-left: 10px;
     padding-right: 10px;
     margin-top: 10px;
}
```

脚本 12-15 这是绘制柱状图的代码

```
window.addEventListener("load",initAll,false);

function initAll() {
     var radioButtons = document.getElementsByTagName("input");

     for (var i=0; i<radioButtons.length;i++) {
        if (radioButtons[i].type == "radio") {
           radioButtons[i].addEventListener("click",chgChart,false);
        }
     }
     chgChart();
}

function chgChart() {
     var bChart = {
        name: "Desktop browser usage by year",
        years: [2010,2011,2012,2013,2014],
        fieldnames: ["MS IE","Firefox","Chrome"],
        fields: [
           [51.45,42.93,33.74,29.25,24.5],
           [31.27,28.2,24.15,20.82,20.53],
           [10.25,21.08,33.23,42.63,46.6]
        ]
     }

     var mobiChart = {
        name: "Mobile device vendors by year",
        years: [2010,2011,2012,2013,2014],
        fieldnames: ["Nokia","Apple","Samsung","RIM"],
        fields: [
           [36.93,38.44,29.27,21.4,17.6],
           [28.88,27.51,24.39,24.01,23.23],
           [4.5,11,18.96,25.47,29.39],
           [19.78,14.38,5.22,3.65,2.87]
        ]
     }

     var radioButtons = document.getElementsByTagName("input");
     var currDirection = getButton("direction");
     var imgSrc = "images/" + getButton("color");

     if (getButton("type")=="browser") {
        var thisChart = bChart;
     }
```

12

```
      else {
         var thisChart = mobiChart;
      }

      var chartBody = "<h2>"+thisChart.name+"</h2><table>";

      for (var i=0; i<thisChart.years.length;i++) {
         if (currDirection == "horizontal") {
            chartBody += "<tr class='horiz'><th rowspan="+(thisChart.fieldnames.length+1);
            chartBody += ">"+thisChart.years[i]+"</th><td colspan=2></td></tr>";
            for (var j=0; j<thisChart.fieldnames.length; j++) {
               chartBody += "<tr class='horiz'><td>"+thisChart.fieldnames[j];
               chartBody += "</td><td><img alt='horiz bar' src='"+imgSrc;
               chartBody += "' width="+thisChart.fields[j][i]*3+">";
               chartBody += thisChart.fields[j][i]+"</td></tr>";
            }
         }
         else {
            chartBody += "<tr class='vert'><th rowspan=2>"+thisChart.years[i]+"</th>";
            for (var j=0; j<thisChart.fieldnames.length; j++) {
               chartBody += "<td><img alt='vert bar' src='"+imgSrc;
               chartBody += "' height="+thisChart.fields[j][i]*3+"'></td>";
            }
            chartBody += "</tr><tr class='vert'>";
            for (j=0; j<thisChart.fieldnames.length; j++) {
               chartBody += "<td>"+thisChart.fields[j][i]+"<br>";
               chartBody += thisChart.fieldnames[j]+"</td>";
            }
            chartBody += "</tr>";
         }
      }

      chartBody += "</table>";
      document.getElementById("chartArea").innerHTML = chartBody;

      function getButton(buttonSet) {
         for (var i=0; i<radioButtons.length;i++) {
            if (radioButtons[i].name == buttonSet && radioButtons[i].checked) {
               return radioButtons[i].value;
            }
         }
         return -1;
      }
}
```

⇒ 生成柱状图

1. ```
var radioButtons = document.getElementsByTagName("input");

for (var i=0; i<radioButtons.length; i++) {
  if (radioButtons[i].type == "radio") {
    radioButtons[i].addEventListener("click",chgChart,false);
  }
}
chgChart();
```

首先执行 initAll()函数。在这里，我们获得所有单选按钮并且遍历它们，将它们设置为在单击时调用 chgChart()。在此之后，手动调用 chgChart()来显示页面的默认视图。
```

2. var bChart = {

在 chgChart()中，我们创建了第一个自定义对象 bChart[browser chart（浏览器图表）的缩写]。创建对象就是这么简单。

```
3. name: "Desktop browser usage by year",
   years: [2010,2011,2012,2013,2014],
   fieldnames: ["MS IE","Firefox","Chrome"],
   fields: [
     [51.45,42.93,33.74,29.25,24.5],
     [31.27,28.2,24.15,20.82,20.53],
     [10.25,21.08,33.23,42.63,46.6]
   ]
```

这里创建自定义对象的属性，并且通过赋值对它们进行初始化。在这里，我们建立了 bChart 的 name、years、fieldnames 和 fields 属性。这些字段分别是图表的名称、图表涉及的年份、图表值的 3 个标签以及每个年份和每个标签的值域（在这个示例中，分别表示一种浏览器）。

注意，在每个属性前面没有使用 var 关键字，这是因为它们不是新变量，而是在一个现有变量（即刚才创建的对象）中添加的新属性。

新属性 fields 之所以使用两层方括号，是因为它是一个二维数组。我们可以用 bChart.fields[0][n]引用第一行，用 bChart.fields[1][n]引用第二行，用 bChart.fields[2][n]引用第三行。

4. }

闭花括号表示完成创建 bChart 对象。

```
5. var mobiChart = {
     name: "Mobile device vendors by year",
     years: [2010,2011,2012,2013,2014],
     fieldnames: ["Nokia","Apple","Samsung","RIM"],
     fields: [
       [36.93,38.44,29.27,21.4,17.6],
       [28.88,27.51,24.39,24.01,23.23],
       [4.5,11,18.96,25.47,29.39],
       [19.78,14.38,5.22,3.65,2.87]
     ]
   }
```

现在，按照创建 bChart 对象的方式，创建 mobiChart[Mobile device chart（移动设备图表）的缩写]对象并且分配它的属性。对于移动设备图表，也需要年份，但是这一次显示的数据是供应商市场份额的百分比。

```
6. var radioButtons = document.getElementsByTagName("input");
   var currDirection = getButton("direction");
   var imgSrc = "images/" + getButton("color");
```

在绘制图表之前，需要知道用户选择了哪个单选按钮。radioButtons 数组包含页面上的所有输入

元素，获得这个数组之后，就可以调用 getButton()函数。将一个字符串（单选按钮集的名称）传递给 getButton()函数，它就会返回一个字符串（这个单选按钮集的当前值）。

可以在 getButton()中建立 radioButtons 数组，而不是在这里。但是采用现在这种方式，就只需要对它进行一次初始化，而不是三次（每次调用 getButton()时执行一次初始化）。

```
7. if (getButton("type")=="browser") {
     var thisChart = bChart;
   }
   else {
     var thisChart = mobiChart;
   }
```

当用户通过单击任何单选按钮来改变图表时，调用 chgChart()函数。当发生这种情况时，如果需要浏览器图表，那么将整个 bChart 对象存储在 thisChart 中。否则，就要显示 JavaScript 图表，所以将 thisChart 赋值为 mobiChart 对象。

```
8. var chartBody = "<h2>"+thisChart.name+"</h2><table>";
```

实际的绘制代码从这里开始。首先，写出图表的名称（存储在 thisChart.name 中，显示在一个<h2>标签中），然后开始一个<table>标签。从这里开始，将在 chartBody 变量中添加内容，最后将把它写到页面上。

```
9. for (var i=0; i<thisChart.years.length; i++) {
```

这是双层循环的第一层，我们用这两个循环遍历步骤 3 中建立的二维数组。这个外层循环使用 i 作为索引变量，循环的次数是图表涉及的年份数量。

```
10. if (currDirection=="horizontal") {
```

如果用户希望看到图表的水平版本（见图 12-8），就运行以下代码。

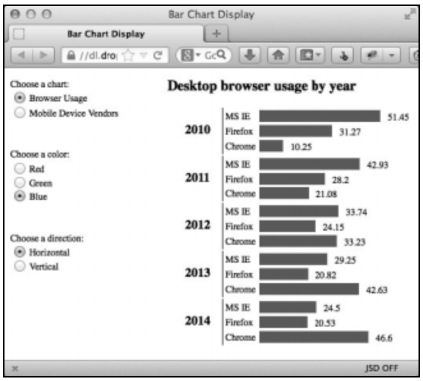


图 12-8　柱状图的初始水平版本

```
11. chartBody += "<tr class='horiz'><th rowspan="+(thisChart.fieldnames.length+1);
    chartBody += ">"+thisChart.years[i]+"</th><td colspan=2></td></tr>";
```

每个水平图表的第一行包含第 i 个年份标签。因为两个图表的字段数可能不一样，所以我们需要实时计算 rowspan。

12.
```
for (var j=0; j<thisChart.fieldnames.length; j++) {
```

这是内层循环的水平版本。这个内层循环使用 j 作为索引变量，循环的次数是存储的 fieldnames 数组的元素数量。

13.
```
chartBody += "<tr class='horiz'> <td>"+thisChart.fieldnames[j];
```

表格的细节行从这里开始。首先写出值标签（浏览器类型或 JavaScript 版本），此数据存储在 fieldnames 数组的第 j 个元素中。

14.
```
chartBody += "</td><td><img alt='horiz bar' src='"+imgSrc;
  chartBody += "' width="+thisChart.fields[j][i]*3+">";
```

接下来,结束前面的单元格并且计算柱状图像。柱状图像的颜色取决于 imgSrc,宽度是数组中[j][i]元素的值乘以 3。例如，如果 imgSrc 是 lilBlue.gif，thisChart. fields[3][4]是 30，那么形成的图像标签会绘制一个蓝色的宽为 90 像素的矩形。

15.
```
chartBody += thisChart.fields[j][i]+"</td></tr>";
```

现在，在图像的右边写出实际数据值，结束这一行。水平图表代码的内层循环到这里就结束了。

16.
```
chartBody += "<tr class='vert'><th rowspan=2>"+thisChart.years[i]+"</th>";
```

如果用户希望看到图表的垂直版本（见图 12-9），那么首先写图表的初始行。图表的垂直版本要稍微复杂一些，而且需要两个独立的内层 j 循环。这里写出图表的标签。

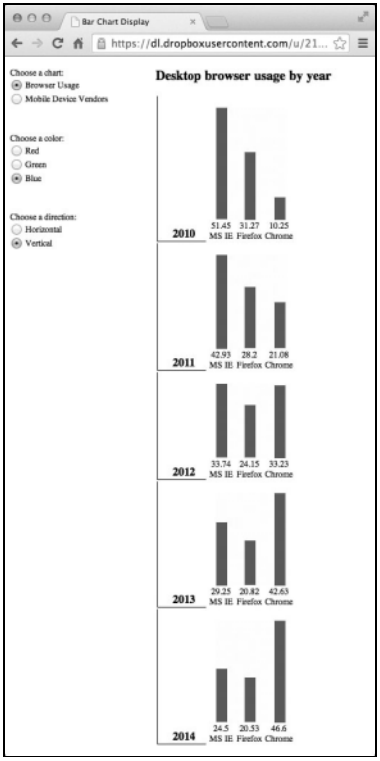


图 12-9　柱状图的垂直版本

12

17. `for (var j=0; j<thisChart.fieldnames.length; j++) {`

这里是第一个内层循环。它在一行中绘制图上的所有线条。

18.
```
chartBody += "<td><img alt='vert bar' src='"+imgSrc;
chartBody += "' height="+thisChart.fields[j][i]*3+"></td>";
```

这里动态地编写图像标签。这一次，高度值基于二维数组中的值。例如，如果 `imgSrc` 是 `lilGreen.gif`，`thisChart.fields[3][4]`是 30，那么形成的图像标签会绘制一个绿色的矩形，高为 90 像素。

19. `chartBody += "</tr><tr class='vert'>";`

绘制完图上的所有线条之后，结束这个表格行并开始一个新行。

20. `for (j=0; j<thisChart.fieldnames.length; j++) {`

这是第二个内层循环。它在对应的线条下面写出每个数据点的值，然后写出 y 轴的标签。

21.
```
chartBody += "<td>"+thisChart.fields[j][i]+"<br>";
chartBody += thisChart.fieldnames[j]+"</td>";
```

这里写出每个线条的信息。数组元素 `thisChart.fields[j][i]`是这个线条的值，`thisChart.fieldnames[j]`是它的数据标签。

22. `chartBody += "</tr>";`

在结束最后一个内层循环之后，需要结束最后的行标签。

23.
```
chartBody += "</table>";
document.getElementById("chartArea").innerHTML = chartBody;
```

现在，水平和垂直代码都结束了，外层循环也结束了，所以写出最后的表格标签以结束脚本，然后将组成的整个字符串放进页面的 `chartArea` 部分的 `innerHTML` 属性中。

✔提示

- 这里的代码使用 3 个图像：lilRed.gif、lilBlue.gif 和 lilGreen.gif。它们都是具有对应颜色的单像素的 GIF。HTML 允许设置图像的高度和宽度，而不管图像的实际尺寸，所以可以用单像素的图像创建任何大小和形状的线条。
- 可能读者会有疑问，水平栏的高度和垂直栏的宽度都是在脚本 12-14 的 CSS 文件中设置的。由于这些数值是静态的（它们不会改变），因此没有必要使用 JavaScript 进行实时设置。
- 只需修改步骤 3 ~ 5 中的数组值，就可以将这个图表改为显示几乎任何数据。无论将这些数组设置成什么，都不需要修改创建图表的循环。
- 这些图表中的统计数据来自 StatCounter's Global Statistics（gs.statcounter.com）。
- 如果对步骤 3 和步骤 5 中数组的代码感到陌生，请参考脚本 12-10。
- 如果对创建新对象的代码感到陌生，建议回顾脚本 10-11。

12.7　样式表切换器

JavaScript 最强大的用途之一是，在运行时改变页面所使用的样式表。例如，可以向站点的访问者提供样式选项，从而允许他们选择站点上文本的样式和大小。一些人喜欢阅读小的 sans-serif 文本，这样就能够在屏幕上看到大量单词（见图 12-10），而其他人喜欢比较大的 serif 文本，这样可读性更好（见图 12-11）。现在，可以让这两类访问者都满意。甚至更进了一步，这个脚本还使用 cookie 存储用

户的选择，这样用户以后访问时会自动采用他们原来选择的样式。

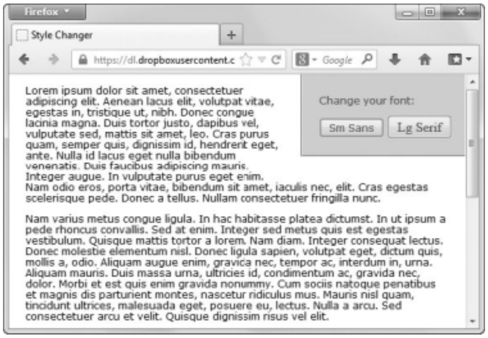


图 12-10　一些访问者喜欢阅读小的 sans-serif 文本，这样就能够在页面上看到更多的文本



图 12-11　其他访问者喜欢选择比较大的 serif 文本，这样可读性更好

脚本 12-16　这个页面包含页面内容和用户控件，它调用用户选择的外部样式表

```
<!DOCTYPE html>
<html>
<head>
     <title>Style Changer</title>
     <link href="script07.css" rel="stylesheet">
     <link href="sansStyle.css" rel="stylesheet" title="default">
     <link href="serifStyle.css" rel="alternate stylesheet" title="serif">
     <script src="script07.js"></script>
</head>
<body>
     <div class="navBar"><p>Change your font:</p>
        <input type="button" id="default" class="sansBtn" value="Sm Sans"
        <input type="button" id="serif" class="serifBtn" value="Lg Serif">
     </div>
     <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean lacus elit, volutpat vitae,
     →egestas in, tristique ut, nibh. Donec congue lacinia magna. Duis tortor justo,dapibus vel,
     →vulputate sed, mattis sit amet, leo. Cras purus quam, semper quis, dignissim id, hendrerit eget,
     →ante. Nulla id lacus eget nulla bibendum venenatis. Duis faucibus adipiscing mauris. Integer augue.
     →In vulputate purus eget enim. Nam odio eros, porta vitae,bibendum sit amet, iaculis nec, elit. Cras
     →egestas scelerisque pede. Donec a tellus. Nullam consectetuer fringilla nunc.</p>

     <p>Nam varius metus congue ligula. In hac habitasse platea dictumst. In ut ipsum a pede rhoncus
     →convallis. Sed at enim. Integer sed metus quis est egestas vestibulum. Quisque mattis tortor
     →a lorem. Nam diam. Integer consequat lectus. Donec molestie elementum nisl. Donec ligula sapien,
     →volutpat eget, dictum quis, mollis a, odio. Aliquam augue enim, gravida nec, tempor ac, interdum
     →in, urna. Aliquam mauris. Duis massa urna, ultricies id,condimentum ac, gravida nec, dolor.
     →Morbi et est quis enim gravida nonummy. Cum sociis natoque penatibus et magnis dis parturient
     →montes, nascetur ridiculus mus. Mauris nisl quam,tincidunt ultrices, malesuada eget, posuere
     →eu, lectus. Nulla a arcu. Sed consectetuer arcu et velit. Quisque dignissim risus vel elit.</p>

     <p>Nunc massa mauris, dictum id, suscipit non, accumsan et, lorem. Suspendisse non lorem quis dui
     →rutrum vestibulum. Quisque mauris. Curabitur auctor nibh non enim. Praesent tempor aliquam
     →ligula. Fusce eu purus. Vivamus ac enim eget urna pulvinar bibendum. Integer porttitor, augue
     →et auctor volutpat, lectus dolor sagittis ipsum, sed posuere lacus pede eget wisi. Proin vel
     →arcu ac velit porttitor pellentesque. Maecenas mattis velit scelerisque tellus. Cras eu tellus
     →quis sapien malesuada porta. Nunc nulla. Nullam dapibus malesuada lorem. Duis eleifend rutrum
```

```
    ➝tellus. In tempor tristique neque. Mauris rhoncus. Aliquam purus.</p>

    <p>Morbi felis quam, placerat sed, gravida a, bibendum a, mauris. Aliquam porta diam. Nam consequat
    ➝feugiat diam. Fusce luctus, felis ut gravida mattis, ante mi viverra sapien, a vestibulum tellus
    ➝lectus ut massa. Duis placerat. Aliquam molestie tellus. Suspendisse potenti. Fusce aliquot
    ➝tellus a lectus. Proin augue diam, sollicitudin eget, hendrerit non, semper at, arcu. Sed
    ➝suscipit tincidunt nibh. Donec ullamcorper. Nullam faucibus euismod augue. Cras lacinia. Aenean
    ➝scelerisque, lorem sed gravida varius, nunc tortor gravida odio, sed sollicitudin pede augue
    ➝ut metus. Maecenas condimentum ipsum et enim. Sed nulla. Ut neque elit, varius a, blandit quis,
    ➝facilisis sed, velit. Suspendisse aliquam odio sed nibh.</p>
</body>
</html>
```

脚本 12-17 这个 CSS 包含总是加载的样式，无论选择哪种字体

```
body {
    margin: 0 20px;
    padding: 0;
    background-color: #FFF;
    color: #000;
}

div.navBar {
    background-color: #CCC;
    width: 175px;
    position: relative;
    top: -20px;
    right: -20px;
    float: right;
    padding: 20px 0 20px 20px;
    border-left: 2px groove #999;
    border-bottom: 2px groove #999;
}

.sansBtn {
    font: 12px/13px verdana, geneva, arial, helvetica, sans-serif;
}

.serifBtn {
    font: 16px/17px "Times New Roman", Times, serif;
}
```

脚本 12-18 这个样式表（sansStyle.css）将所有文本改为小的 sans-serif 字体

```
body, p, td, ol, ul, select, span, div, input {
  font: 12px/13px verdana, geneva, arial, helvetica, sans-serif;
}
```

脚本 12-19 这个样式表（serifStyle.css）将页面文本改为比较大的 serif 字体

```
body, p, td, ol, ul, select, span, div, input {
    font: 16px/17px "Times New Roman",Times, serif;
}
```

脚本 12-20 这个脚本处理当前样式表的设置

```
window.addEventListener("load",initStyle,false);
window.addEventListener("unload",unloadStyle,false);

function initStyle() {
    var thisCookie = cookieVal("style");
```

```
    if (thisCookie) {
       var title = thisCookie;
    }
    else {
       var title = getPreferredStylesheet();
    }
    setActiveStylesheet(title);

    var allButtons = document.getElementsByTagName("input");
    for (var i=0; i<allButtons.length; i++) {
       if (allButtons[i].type == "button") {
          allButtons[i].addEventListener("click",setActiveStylesheet,false);
       }
    }
}

function unloadStyle() {
    var expireDate = new Date();
    expireDate.setYear(expireDate.getFullYear()+1);
    document.cookie = "style=" + getActiveStylesheet() + ";expires=" + expireDate.toGMTString() +
    ➝";path=/";
}

function getPreferredStylesheet() {
    var thisLink, relAttribute;
    var linksFound = document.getElementsByTagName("link");

    for (var i=0; i<linksFound.length; i++) {
       thisLink = linksFound[i];
       relAttribute = thisLink.getAttribute("rel");
       if (relAttribute.indexOf("style") > -1 && relAttribute.indexOf("alternate") == -1 && thisLink.
       ➝getAttribute("title")) {
          return thisLink.getAttribute("title");
       }
    }
    return "";
}

function getActiveStylesheet() {
    var thisLink, relAttribute;
    var linksFound = document.getElementsByTagName("link");

    for (var i=0; i<linksFound.length; i++) {
       thisLink = linksFound[i];
       relAttribute = thisLink.getAttribute("rel");
       if (relAttribute.indexOf("style") > -1 && thisLink.getAttribute("title") && !thisLink.disabled) {
          return thisLink.getAttribute("title");
       }
    }
    return "";
}

function setActiveStylesheet(inVal) {
    var thisLink, relAttribute;
    var linksFound = document.getElementsByTagName("link");

    if (inVal) {
       if (typeof inVal == "string") {
          var title = inVal;
       }
```

```
            else {
               var title = inVal.target.id;
            }
         }
         else {
            var title = window.event.srcElement.id;
         }

         for (var i=0; i<linksFound.length; i++) {
            thisLink = linksFound[i];
            relAttribute = thisLink.getAttribute("rel");
            if (relAttribute.indexOf("style") > -1 && thisLink.getAttribute("title")) {
               if (thisLink.getAttribute("title") == title) {
                  thisLink.disabled = false;
               }
               else {
                  thisLink.disabled = true;
               }
            }
         }
      }

      function cookieVal(cookieName) {
         var thisCookie = document.cookie.split("; ");
         for (var i=0; i<thisCookie.length; i++) {
            if (cookieName == thisCookie[i].split("=")[0]) {
               return thisCookie[i].split("=")[1];
            }
         }
         return "";
      }
```

⇛ 允许用户切换样式表

1. `<link href="sansStyle.css" rel="stylesheet" title="default">`

脚本 12-16 包含一个到外部样式表的标准链接，但是其中多了一点新东西：它有一个 title 属性 default。这会在后面的操作中起作用。

2. `<link href="serifStyle.css" rel="alternate stylesheet" title="serif">`

这里使用 link 标签链接另一个样式表。但是，rel 属性没有设置为通常的 stylesheet，而是设置为 alternate stylesheet。这是因为这个样式表实际上不是默认样式表，而是只在用户选择它时使用。

3.
```
<input type="button" id="default" class="sansBtn" value="Sm Sans">
<input type="button" id="serif" class="serifBtn" value="Lg Serif" >
```

这是两个按钮：Sm Sans 和 Lg Serif。单击前一个按钮，就会使页面上的所有文本显示为小的 sans-serif 字体；单击后一个按钮，就会使页面上的所有文本显示为比较大的 serif 字体。如果浏览器支持的话，脚本 12-17 中的样式会使按钮文本本身显示为目标字体，这使用户能够预先看到选择这个按钮之后的文本效果。

4.
```
body, p, td, ol, ul, select, span, div, input {
   font: 12px/13px verdana, geneva, arial, helvetica, sans-serif;
}
```

脚本 12-18（sansStyle.css）告诉浏览器，在装载它时，它涉及的所有标签都应该显示为 12px 的 Verdana 字体（或者用户计算机上安装的某种 sans-serif 字体）。

```
5. body, p, td, ol, ul, select, span, div, input {
     font: 16px/17px "Times New Roman", Times, serif;
   }
```

与之相反，脚本 12-19（serifStyle.css）告诉浏览器，它涉及的所有标签都应该显示为 16px 的 Times New Roman 字体（或者可以找到的其他某种 serif 字体）。

```
6. var thisCookie = cookieVal ("style");
   if (thisCookie) {
     var title = thisCookie;
   }
   else {
     var title = getPreferredStylesheet();
   }
   setActiveStylesheet(title);
```

脚本 12-20 中的 initStyle()函数在页面运行时加载，它的目标是对页面需要的所有东西进行初始化。在这里，检查这个用户是否已经设置了 cookie（cookie 中保存他选择的样式）。我们的老伙伴 cookieVal()函数来自第 9 章，它读取 cookie 并且检查是否有称为 style 的 cookie。如果有，它的值就是我们需要的样式表；如果没有，就调用 getPreferredStylesheet()。知道了所需的样式表之后，就调用 setActiveStylesheet()来设置页面外观。

```
7. var allButtons = document.getElementsByTagName("input");
   for (var i=0; i<allButtons.length;i++) {
     if (allButtons[i].type == "button") {
       allButtons[i].addEventListener("click",setActiveStylesheet,false);
     }
   }
```

initStyle()函数还需要给按钮添加事件处理程序。在这里，在单击按钮时，事件处理程序都调用 setActiveStylesheet()。

12

```
8. function unloadStyle() {
     var expireDate = new Date();
     expireDate.setYear(expireDate.getFullYear()+1);
     document.cookie = "style=" + getActiveStylesheet() + ";expires=" + expireDate.toGMTString()
     ➝+ ";path=/";
   }
```

当页面卸载时，需要设置 cookie 供以后使用。cookie 的终止日期设置为一年之后，然后调用 getActiveStylesheet()获得用户当前的设置，并且写出 cookie 供以后使用。

```
9. function getPreferredStylesheet() {
     var thisLink, relAttribute;
     var linksFound = document.getElementsByTagName("link");
```

如果在加载页面时，没有 cookie 能够提供用户以前选择的样式，脚本就需要判断首选的样式表。

这个步骤和下一个步骤中的 getPreferredStylesheet()实现这个功能。

10.
```
for (var i=0; i<linksFound.length;i++) {
    thisLink = linksFound[i];
    relAttribute = thisLink.getAttribute("rel");
    if (relAttribute.indexOf("style") > -1 && relAttribute.indexOf("alternate") == -1 &&
    ➝thisLink.getAttribute("title")) {
      return thisLink.getAttribute("title");
    }
  }
```

这个函数遍历每个链接标签，检查它们是否有 rel 属性，这个属性的值是否包含 style，这个属性的值是否不包含 alternate，以及标签是否有 title 属性。如果找到符合所有这些条件的链接标签，它链接的就是首选样式表，所以返回它的 title 属性。

为了了解代码中的哪个实际标签是首选样式表，看一下 HTML 文件中的 link 标签。虽然 HTML 文件中有 3 个 link 标签，但是只有两个有 title 属性。在这两个 link 标签中，一个有 rel 属性 stylesheet，另一个的 rel 属性是 alternate stylesheet。因此，首选样式表是 title 属性为 default 的那个。

11.
```
for (var i=0; i<linksFound.length; i++) {
    thisLink = linksFound[i];
    relAttribute = thisLink.getAttribute("rel");
    if (relAttribute.indexOf("style") > -1 && thisLink.getAttribute ("title")&& !thisLink.
    ➝disabled) {
      return thisLink.getAttribute("title");
    }
  }
```

正如前面提到的，当用户离开这个站点时，我们希望使用 cookie 存储他们选择的样式表。这样的话，当他们以后再次访问这里时，就会看到他们喜欢的字体。尽管可以在用户每次单击样式按钮时写 cookie，但更好的方法是只在用户离开站点时写 cookie。在这里，getActiveStylesheet()函数（当页面卸载时会调用它）循环遍历所有 link 标签，选择当前启用的样式表，并且返回这个样式表的 title。

12.
```
var thisLink, relAttribute;
  var linksFound = document.getElementsByTagName("link");

  if (inVal) {
    if (typeof inVal == "string") {
      var title = inVal;
    }
    else {
      var title = inVal.target.id;
    }
```

```
    }
    else {
      var title = window.event.srcElement.id;
    }
```

正如前面看到的，当用户加载这个页面时，调用 `setActiveStylesheet()`函数并且传递一个参数，这个参数在函数内称为 `inVal`。但是，在单击按钮之后也会调用 `setActiveStylesheet()`。在这种情况下，根据使用的浏览器和浏览器处理事件的方式，可能传递参数，也可能不传递。在这里进行检查，从而判断哪个函数调用了这个函数以及用户希望的操作。有以下 3 种可能的情况。

- ❑ `initStyle()`调用了这个函数，并且传递给它一个包含首选样式表的字符串。在这种情况下，`inVal` 存在而且它是一个字符串，所以将 `title` 设置为 `inVal`。
- ❑ 在支持 W3C 样式事件的浏览器中单击了样式按钮。在这种情况下，`inVal` 自动设置为触发此函数的事件，所以 `inVal` 存在，但它不是字符串。当发生这种情况时，事件的 `target`（即导致触发事件的目标）就是被单击的按钮，这个按钮的 `id` 存储着所需的样式名称。
- ❑ 在不支持 W3C 标准但支持 IE 事件模型的浏览器中单击了样式按钮。如果是这种情况，那么 `inVal` 变量不存在，所以要从 `window.event.srcElement.id` 获得所需的样式。

13.
```
thisLink = linksFound[i];
relAttribute = thisLink.getAttribute("rel");
if (relAttribute.indexOf("style") > -1 && thisLink.getAttribute("title")) {
  if (thisLink.getAttribute("title") == title) {
    thisLink.disabled = false;
  }
  else {
    thisLink.disabled = true;
  }
}
```

`setActiveStylesheet()`函数遍历文档中的所有链接标签，检查每个标签是否同时具有包含 style 的 `rel` 属性和 `title` 属性。如果这两个条件都成立，那么首先禁用这个链接，然后在并且只在 `title` 属性设置为 `title` 值的情况下重新启用它。

所以，如果当前使用的样式表的 `title` 属性为 default，而用户单击了 Lg Serif 按钮，JavaScript 就知道应该装载 `serif` 样式表。有一个 `link` 标签具有 `title` 属性 serif，所以禁用所有其他样式表（即这里的 `default` 样式表），只打开 `serif` 样式表。

12

第 13 章

Ajax 简介

13

Web 总是在不断变化，Web 和 JavaScript 的大趋势在 2005 年年初又有了新变化。出现了新型的 Web 应用程序，而且它们很快流行起来了，这些应用程序包括来自谷歌的 Gmail 和 Google Maps，以及来自其他公司的 Flickr 等程序。这些新站点的共同特色是，它们表现得更像桌面应用程序，具有快速的高响应性的用户界面。在传统的 Web 应用程序中，当用户单击链接时，要等待服务器作出响应并且刷新页面，而且这个过程会重复进行。但是，这些新站点具有更好的响应性，能够立即更新页面，这提供了出色的交互和更好的用户体验。

这些新站点的强大功能来自于称为 Ajax 的新技术（其实这种技术并不是全新的）。可以使用 Ajax 技术让自己的站点具有更好的响应性、更吸引人，这会使站点的用户在浏览过程中更愉快。更棒的是，你不需要学习全新的技术，因为 Ajax 是由你已经掌握的几种技术组合而成的（本书前面已经讨论过这些技术）。

在本章中，你将学习如何在后台向服务器请求信息，并且将它转换为 Ajax 应用程序可以使用的形式，自动刷新来自服务器的信息，为页面上的对象构建很酷的预览效果，以及构建一个 Ajax 应用程序，它能够像桌面应用程序那样自动补全表单字段。

13.1　Ajax 的定义

关于 Ajax 的有趣现象之一是，对于 Ajax 究竟是什么有一些混淆，甚至是争论。我们知道 Ajax 很重要而且非常流行，甚至因此在本书前几版的书名中增加了 Ajax 这个词，以此迎合 Ajax 潮流。所以有必要澄清 Ajax 是什么，以及在我们使用这个术语时指的是什么。

首先，谈谈 Ajax 的历史。2005 年 2 月，Jesse James Garrett（Adaptive Path 的创始人之一，Adaptive Path 是美国旧金山的一个 Web 界面和设计公司）在他们站点上的一篇文章中首次提出了 Ajax 这个术语。他说 Ajax 是 Asynchronous JavaScript and XML（异步 JavaScript 和 XML）的缩写（但不是首字母缩写）。在 adaptivepath.com/ideas/ajax-new-approach-web-applications/上可以读到这篇文章（见图 13-1）。



图 13-1　这是启动 Ajax 热潮的那篇文章

根据 Garrett 的说法，Ajax 本身并不是一种新技术，它是由几种长期存在的 Web 技术组合而成的：

- 使用 HTML 和 CSS 控制页面结构和表示方式；
- 使用 DOM 显示和操纵页面；
- 使用浏览器的 `XMLHttpRequest` 对象在客户机和服务器之间传输数据①；
- 使用 XML 作为在客户机和服务器之间传输的数据的格式②；
- 最后，使用 JavaScript 动态地显示所有内容并且提供交互功能。

Ajax 应用程序在用户和服务器之间建立一个中介。Ajax 引擎（Ajax engine，也称为网页的 JavaScript 部分）向用户提供界面（当然要借助于 HTML 和 CSS）。如果用户的操作并不要求向服务器发出请求（例如，显示已经存储在本地的数据），那么 Ajax 引擎会进行响应。这使浏览器能够对许多用户操作立刻作出反应，使网页的反应像桌面程序那样迅速。如果用户操作需要服务器调用，Ajax 引擎就异步地执行它，因此用户不需要等待服务器的响应。用户可以继续与应用程序进行交互，当请求的数据到达时，引擎会更新页面。这里的重点是，用户的操作不会由于等待服务器而暂停。

随着这种技术的发展，即使 Web 程序不包含所有这些组成部分，也可以称为 Ajax 应用程序，由此引起了混淆和争论。实际上，甚至本书的两位作者对此也有分歧。

13

① 某些场合也可以使用 iframe，或者动态添加`<script>`标签。——编者注

② 也可以使用 JSON 等格式。——编者注

Tom 说："我喜欢只用 DOM、HTML、CSS 和 JavaScript 操作页面，我把这种方式称为 Ajax。人们把许多效果都称为 Ajax，而且现代站点的整个外观由于 Ajax 而改变了。从静态网页到动态网页（有时候称为 Web 2.0）的改变，在外观和感觉方面借助于 Ajax 技术，无论在幕后是否有服务器调用。这样宽泛地定义 Ajax 可能会让纯粹主义者不满意，但是我认为是合适的。"

而另一位作者 Dori 是正统的 JavaScript 程序员，她认为："要想成为 Ajax 应用程序，就需要在客户机和服务器之间传递一些数据。否则，怎么算是新技术呢？"

Dori 正在编写 Ajax 代码，所以对于本章的大多数部分，我们采用她对 Ajax 应用程序是什么及其应该做什么的看法。但在其他章节中，我们将演示如何在站点上添加一些 Web 2.0 风格的特性，这些特性具有实际作用，而不只是装饰。

现在，我们谈谈什么东西不属于 Ajax 的范围。因为可以使用 Ajax 在网页上实现某些很酷的视觉效果，有些人认为能够使页面看起来更好的任何东西都是 Ajax，这导致他们把 Flash 建立的界面也称为 Ajax。但是，这是一种误解。Ajax 的作用并不是把好看的用户界面组件放在站点上，使用户界面更酷，而是改变用户操作网页的习惯。

那么 Ajax 会存在一些问题，这些问题很重要。例如，要想正确地工作，Ajax 站点需要运行在现代浏览器中。它还需要 JavaScript。所以，对于使用老式浏览器或者关闭了 JavaScript 功能的用户，应该怎么做呢？对于残障用户或者使用功能有限的手持设备（比如手机或手写板）的用户，又应该怎么做呢？答案是，必须让站点能够平稳地退化，这意味着使用功能不足的浏览器的用户可以使用站点功能的一部分，至少应该显示有意义的错误消息，让他们知道为什么不能使用你的站点。

Ajax 应用程序的另一个潜在问题是，它们可能破坏浏览器后退按钮的正常表现。对于静态页面，在单击后退按钮时，用户会期望浏览器转到它加载的前一个页面。但是，因为启用 Ajax 的页面是动态更新的，所以这种期望可能会落空。对于"后退按钮问题"，有一些解决方案，在全面投入 Ajax 开发之前，你应该考虑这个问题及其解决方案。

另外，Ajax 不依赖于特定的服务器端技术。有许多公司试图借助于 Ajax 热潮推销他们的服务器端解决方案，这只是他们做生意的手段，但是没有理由非用他们的产品不可。只要后端能够提供 JavaScript 可以读取的信息（XML 形式），Ajax 就能够运行。那些公司（比如 IBM）用华丽的辞藻宣传他们的产品，只是为了让产品搭上 Ajax 的顺风车，让你购买，你大可不必相信。

13.2　读取服务器数据

我们从基础开始讲解 Ajax：使用 `XMLHttpRequest` 对象获得和显示来自服务器的数据。

为了完成这个任务，我们将使用脚本 13-1（HTML）和脚本 13-2（JavaScript）。可以读取的文件有两个：脚本 13-3 所示的纯文本文件和脚本 13-4 所示的 XML 文件。

脚本 13-1　文本文件和 XML 文件请求示例的 HTML

```
<!DOCTYPE html>
<html>
<head>
     <title>My First Ajax Script</title>
     <script src="script01.js"></script>
</head>
<body>
```

```
    <p>
       <a id="makeTextRequest" href="gAddress.txt">Request a text file</a><br>
       <a id="makeXMLRequest" href="us-states.xml">Request an XML file</a>
    </p>
    <div id="updateArea"> </div>
</body>
</html>
```

脚本 13-2 这个 JavaScript 脚本从服务器获得文件

```
window.addEventListener("load", initAll, false);
var xhr = false;

function initAll() {
     document.getElementById("makeTextRequest").addEventListener("click", getNewFile, false);
     document.getElementById("makeXMLRequest").addEventListener("click", getNewFile, false);
}

function getNewFile(evt) {
     makeRequest(this.href);
     evt.preventDefault();
}

function makeRequest(url) {
     if (window.XMLHttpRequest) {
        xhr = new XMLHttpRequest();
     }
     else {
        if (window.ActiveXObject) {
           try {
              xhr = new ActiveXObject("Microsoft.XMLHTTP");
           }
           catch (e) {
           }
        }
     }

     if (xhr) {
        xhr.addEventListener("readystatechange",showContents,false);
        xhr.open("GET", url, true);
        xhr.send(null);
     }
     else {
        document.getElementById("updateArea").innerHTML = "Sorry, but I couldn't create an XMLHttpRequest";
     }
}

function showContents() {
     if (xhr.readyState == 4) {
        if (xhr.status == 200) {
           if (xhr.responseXML && xhr.responseXML.childNodes.length > 0) {
              var outMsg = getText(xhr.responseXML.getElementsByTagName ("choices")[0]);
           }
           else {
              var outMsg = xhr.responseText;
           }
        }
        else {
           var outMsg = "There was a problem with the request " + xhr.status;
        }
        document.getElementById("updateArea").innerHTML = outMsg;
```

```
    }

    function getText(inVal) {
       if (inVal.textContent) {
          return inVal.textContent;
       }
       return inVal.text;
    }
}
```

脚本 13-3 这是请求的文本文件

```
Four score and seven years ago our fathers brought forth on this continent, a new nation, conceived in Liberty,
➝and dedicated to the proposition that all men are created equal.

Now we are engaged in a great civil war,testing whether that nation, or any nation so conceived and so dedicated,
➝can long endure. We are met on a great battle-field of that war. We have come to dedicate a portion of that
➝field, as a final resting place for those who here gave their lives that that nation might live. It is altogether
➝fitting and proper that we should do this.

But, in a larger sense, we can not dedicate -- we can not consecrate -- we can not hallow -- this ground. The
➝brave men,living and dead, who struggled here, have consecrated it, far above our poor power to add or detract.
➝The world will little note, nor long remember what we say here,but it can never forget what they did here.
➝It is for us the living, rather,to be dedicated here to the unfinished work which they who fought here have
➝thus far so nobly advanced. It is rather for us to be here dedicated to the great task remaining before us
➝-- that from these honored dead we take increased devotion to that cause for which they gave the last full
➝measure of devotion -- that we here highly resolve that these dead shall not have died in vain -- that this
➝nation, under God, shall have a new birth of freedom -- and that government of the people, by the people,
➝for the people,shall not perish from the earth.
```

脚本 13-4 这是请求的 XML 文件

```
<?xml version="1.0"?>
<choices xml:lang="EN">
  <item><label>Alabama</label><value>AL</value></item>
  <item><label>Alaska</label><value>AK</value></item>
  <item><label>Arizona</label><value>AZ</value></item>
  <item><label>Arkansas</label><value>AR</value></item>
  <item><label>California</label><value>CA</value></item>
  <item><label>Colorado</label><value>CO</value></item>
  <item><label>Connecticut</label><value>CT</value></item>
  <item><label>Delaware</label><value>DE</value></item>
  <item><label>Florida</label><value>FL</value></item>
  <item><label>Georgia</label><value>GA</value></item>
  <item><label>Hawaii</label><value>HI</value></item>
  <item><label>Idaho</label><value>ID</value></item>
  <item><label>Illinois</label><value>IL</value></item>
  <item><label>Indiana</label><value>IN</value></item>
  <item><label>Iowa</label><value>IA</value></item>
  <item><label>Kansas</label><value>KS</value></item>
  <item><label>Kentucky</label><value>KY</value></item>
  <item><label>Louisiana</label><value>LA</value></item>
  <item><label>Maine</label><value>ME</value></item>
  <item><label>Maryland</label><value>MD</value></item>
  <item><label>Massachusetts</label><value>MA</value></item>
  <item><label>Michigan</label><value>MI</value></item>
  <item><label>Minnesota</label><value>MN</value></item>
  <item><label>Mississippi</label><value>MS</value></item>
  <item><label>Missouri</label><value>MO</value></item>
  <item><label>Montana</label><value>MT</value></item>
```

```
  <item><label>Nebraska</label><value>NE</value></item>
  <item><label>Nevada</label><value>NV</value></item>
  <item><label>New Hampshire</label><value>NH</value></item>
  <item><label>New Jersey</label><value>NJ</value></item>
  <item><label>New Mexico</label><value>NM</value></item>
  <item><label>New York</label><value>NY</value></item>
  <item><label>North Carolina</label><value>NC</value></item>
  <item><label>North Dakota</label><value>ND</value></item>
  <item><label>Ohio</label><value>OH</value></item>
  <item><label>Oklahoma</label><value>OK</value></item>
  <item><label>Oregon</label><value>OR</value></item>
  <item><label>Pennsylvania</label><value>PA</value></item>
  <item><label>Rhode Island</label><value>RI</value></item>
  <item><label>South Carolina</label><value>SC</value></item>
  <item><label>South Dakota</label><value>SD</value></item>
  <item><label>Tennessee</label><value>TN</value></item>
  <item><label>Texas</label><value>TX</value></item>
  <item><label>Utah</label><value>UT</value></item>
  <item><label>Vermont</label><value>VT</value></item>
  <item><label>Virginia</label><value>VA</value></item>
  <item><label>Washington</label><value>WA</value></item>
  <item><label>West Virginia</label><value>WV</value></item>
  <item><label>Wisconsin</label><value>WI</value></item>
  <item><label>Wyoming</label><value>WY</value></item>
</choices>
```

⇛ 请求服务器数据

1. `var xhr = false;`

脚本 13-2 中的 xhr 变量是在本章中会经常看到的一个变量。它是一个 XMLHttpRequest 对象（或者说在初始化之后将成为 XMLHttpRequest 对象）。目前，我们只需在任何函数之外创建它，使它成为全局变量。

2.
```
function initAll() {
    document.getElementById("makeTextRequest").addEventListener("click",getNewFile,false);
    document.getElementById("makeXMLRequest").addEventListener("click",getNewFile,false);
  }
```

当加载页面时，会调用 initAll()函数。在这里，我们设置两个 click 处理程序，当用户单击这两个链接时触发 getNewFile()函数。

3.
```
function getNewFile(evt) {
    makeRequest(this.href);
    evt.preventDefault();
  }
```

当用户单击链接时，要执行某些操作。在这个示例中，操作是调用 makeRequest()——但是这个函数需要知道已经请求了哪个文件。我们知道 this.href 中存储着这一信息，所以可以传递这个属性。当函数返回时，操作就完成了，所以告诉浏览器我们不希望加载新的网页。

4.
```
if (window.XMLHttpRequest) {
    xhr = new XMLHttpRequest();
  }
```

这些代码在 makeRequest()中，这是有意思的地方。现代浏览器支持一个本机 XMLHttpRequest 对

象（见表 13-1），这个对象是 window 的一个属性。所以，我们检查这个属性是否存在，如果存在，就创建一个新的 XMLHttpRequest 对象。

表13-1 XMLHttpRequest对象

属　性	方　法	事件处理程序
readState	abort	onload
response	addEventListener	onloadend
responseText	getAllResponseHeaders	onloadstart
responseXML	getResponseHeader	onreadystatechange
status	open	ontimeout
statusText	overideMimeType	
timeout	send	
upload	setRequestHeader	
withCredentials		

```
5. if (window.ActiveXObject) {
     try {
       xhr = new ActiveXObject("Microsoft.XMLHTTP");
     }
     catch (e) {
     }
   }
```

虽然微软的 IE（版本 5.5 和版本 6）支持 XMLHttpRequest，但是没有这个对象的本机版本。在这种情况下，必须检查浏览器是否支持 ActiveX。如果支持的话，就检查是否能够根据 ActiveX 创建 XMLHttpRequest 对象（使用 try/catch 异常处理结构）。如果可以，就这么做。

```
6. if (xhr) {
     xhr.addEventListener("readystatechange", showContents, false);
     xhr.open("GET", url, true);
     xhr.send(null);
   }
```

无论采用哪种方式，我们都应该会获得一个新的 xhr 对象。如果获得了 xhr 对象，就需要用它做 3 件事。我们总是用 xhr 做下面 3 件事。

- 设置 xhr 的 readystatechange 事件处理程序。每当 xhr.readyState 属性值发生变化时，就会触发这个处理程序。
- 调用 open()并且传递 3 个参数：一个 HTTP 请求方法（例如"GET"、"POST"或"HEAD"）、服务器上一个文件的 URL 和一个布尔值，这个布尔值告诉服务器请求是否异步（也就是说，我们是否会等待请求完成）。
- 最后，我们用 send()发送刚才创建的请求。如果要请求 POST，就传递这里给出的参数。

```
7. else {
     document.getElementById("updateArea").innerHTML = "Sorry, but I couldn't create an
     →XMLHttpRequest";
   }
```

如果执行到这里，那么就说明由于某种原因无法创建 XMLHttpRequest，所以除了输出错误消息之外，没什么事儿可做了。

8.
```
if (xhr.readyState == 4) {
  if (xhr.status == 200) {
```

现在到了 showContents()函数中。readyState 属性可能是几个值之一（见表 13-2），而且每当服务器改变它的值时，就会触发 showContents()函数。但是，在请求完成之前，我们实际上不希望执行任何操作（至少在这里不做什么），所以首先检查 readyState 是否等于 4。如果是，就可以继续执行，检查请求返回的是什么。

表13-2　readyState属性值

值	意　义
0	未初始化。对象不包含数据
1	正在加载。对象当前正在加载它的数据
2	已经加载。对象已经完成了数据加载
3	交互式的。即使对象还未完全加载完，用户也可能与对象进行交互
4	完成。对象已经完成了初始化

首先要检查的是请求的状态，即服务器返回的状态码（对于请求的每个文件，服务器都会在幕后返回这些编码，但是浏览器只在出现错误时显示它们）。例如，状态码 200 意味着一切正常，这里的状态就是服务器调用返回的状态；如果请求的文件不存在，就会从 Web 服务器得到 404 错误。

9.
```
if (xhr.responseXML && xhr.responseXML.childNodes.length > 0) {
  var outMsg = getText(xhr.responseXML.getElementsByTagName("choices")[0]);
}
else {
  var outMsg = xhr.responseText;
}
```

如果执行到这里，就意味着一切正常，我们要查看服务器实际上提供了什么。我们可以读取的文件有两种类型，所以需要检查返回的数据类型。如果数据是 XML，responseXML 属性就包含数据。然而，responseXML 属性也包含不是 XML 的数据。如果 responseXML.ChildNodes.length 大于零（即它包含多个虚拟对象），那么我们就知道已经得到了一个具有适当格式的 DOM 对象，可以使用以前见过的命令（比如 getElementsByTagName()）来遍历它的节点。但是在这里，我们只想试试那种方法，将其结果传递给 getText()函数。返回的值保存在 outMsg 中。

如果得到的数据不是有效的 XML，那么它就是文本文件。在这种情况下，我们将 xhr 的 responseText 属性放到 outMsg 中。

10.
```
else {
  var outMsg = "There was a problem with the request " + xhr.status;
}
```

如果返回了 200 之外的其他状态码，就是出了问题，所以设置 outMsg 来指出错误，并且附加上状态错误，让用户能够判断出了什么问题。

11. `document.getElementById("updateArea").innerHTML = outMsg;`

最后，将 `outMsg` 输出到屏幕，见图 13-2。

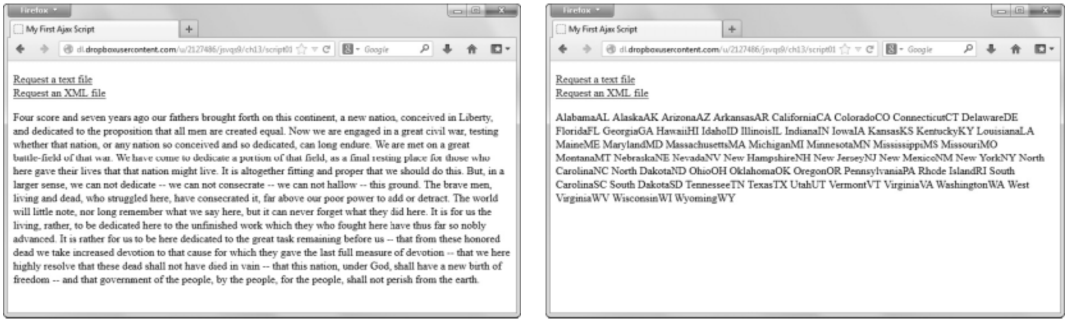


图 13-2　单击链接就会获得葛底斯堡演讲的文本文件（左图）
或者美国各州名称和缩写的 XML 文件（右图）

12.
```
if (inVal.textContent) {
   return inVal.textContent;
  }
  return inVal.text;
```

这就是 `getText()`，其作用是检查所有传入的参数是否包含 `textContent` 属性。如果包含，就返回 `textContent`，否则返回其 `text` 属性。

✔提示

- ❑ 由于 Ajax 的工作方式，在进行开发和测试时，要读取的文件必须驻留在服务器上，它们不能是本地文件。
- ❑ 在步骤 5 中我们说过，IE 5.5 和 IE 6 使用 ActiveX 控件创建 `XMLHttpRequest` 对象。好在 IE 7 有了本机对象，所以不再需要 ActiveX 控件了。但是，这意味着必须先检查本机对象是否存在——如果先检查 `window.ActiveXObject`，那么 IE 7 及以上版本也会通过这个测试，所以会进入错误的代码路径。许多在 IE 7 发布之前编写的 Ajax 代码都有这个问题。
- ❑ 如果需要进一步检查实际获得的微软 ActiveX 对象的版本，那么可以使用下面的代码片段：

```
if (window.ActiveXObject) {
     try {
       xhr = new ActiveXObject("Msxml2.XMLHTTP");
     }
     catch (e) {
       try {
         xhr = new ActiveXObject("Microsoft.XMLHTTP");
       }
       catch (e) { }
     }
```

这个方法首先尝试使用 `XMLHttpRequest` 对象的 IE 6 版本（`Msxml2.XMLHTTP`），如果找不到这个对象版本，就尝试老版本。但是，`Microsoft.XMLHTTP` 应该会提供 PC 上可用的最新版本，所以我们只在本章中使用这样的代码——因为老代码最终会废弃的。

- Ajax 调用的缺点之一是它们可能被缓存。也就是说，应用程序看似与服务器进行通信并且获得新数据，但是它实际上只是查看以前读取的数据。如果是这种情况，设置请求首部会有帮助。添加以下请求首部可以迫使服务器提供最新数据：

```
xhr.setRequestHeader("If-Modified-Since", "Wed, 15 Jan 1995 01:00:00 GMT");
xhr.setRequestHeader("Cache-Control","no-cache");
xhr.setRequestHeader("Cache-Control","must-revalidate");
xhr.setRequestHeader("Cache-Control","no-store");
xhr.setRequestHeader("Pragma","no-cache");
xhr.setRequestHeader("Expires","0");
```

- 通过覆盖 MIME 类型，可以迫使调用返回 XML 数据：

```
xhr.overrideMimeType("text/xml");
```

但是，对于某些浏览器和配置，这可能会造成问题，所以要谨慎使用。

13.3 解析服务器数据

既然已经从服务器获得了数据，我们就需要找到所需的信息，并且确保它采用的格式是 Ajax 应用程序可以使用的。为此，首先需要检查信息。因为数据是结构良好的 XML 文档，脚本可以在 XML 文档树中移动，寻找并且提取所需的数据，然后将数据存储在变量中。如果需要，脚本还可以对数据进行重新格式化以备后期使用。

HTML 和 CSS 很简单，分别见脚本 13-5 和脚本 13-6，所以只需看看 JavaScript 文件（脚本 13-7）中的代码。对于这个示例，XML 文件是关于存储在 Flickr 上的照片的数据。XML 的一部分见脚本 13-8。

脚本 13-5 只需添加一些 JavaScript 代码，这个简单的 HTML 页面就会更引人注目

```
<!DOCTYPE html>
<html>
<head>
     <title>My Second Ajax Script</title>
     <link rel="stylesheet"href="script02.css">
     <script src="script02.js"></script>
</head>
<body>
     <div id="pictureBar"> </div>
</body>
</html>
```

13

脚本 13-6 只需少量 CSS 代码，就足以改善页面的外观

```
img {
    border-width: 0;
    margin: 5px;
}
```

脚本 13-7 这个脚本中增加的 JavaScript 代码可以对前面请求的数据进行解析

```
window.addEventListener("load",initAll,false);
var xhr = false;

function initAll() {
```

```
    if (window.XMLHttpRequest) {
       xhr = new XMLHttpRequest();
    }
    else {
       if (window.ActiveXObject) {
          try {
             xhr = new ActiveXObject("Microsoft.XMLHTTP");
          }
          catch (e) {
          }
       }
    }

    if (xhr) {
       xhr.addEventListener("readystatechange", showPictures, false);
       xhr.open("GET", "flickrfeed.xml", true);
       xhr.send(null);
    }
    else {
       alert("Sorry, but I couldn't create an XMLHttpRequest");
    }
}

function showPictures() {
    var tempText = document.createElement("div");
    var theText

    if (xhr.readyState == 4) {
       if (xhr.status == 200) {
          var allImages = xhr.responseXML.getElementsByTagName("content");

          for (var i=0; i<allImages.length; i++) {
             tempText.innerHTML = getPixVal(allImages[i]);

             theText= tempText.getElementsByTagName("p")[1].innerHTML;
             theText = theText.replace(/240/g,"75");
             theText = theText.replace(/180/g,"75");
             theText = theText.replace(/_m/g,"_s");
             document.getElementById("pictureBar").innerHTML += theText;
          }
       }
       else {
          alert("There was a problem with the request " + xhr.status);
       }
    }

    function getPixVal(inVal) {
       return (inVal.textContent) ? inVal.textContent : inVal.text;
    }
}
```

脚本 13-8　这是 Flickr 提供的 XML 文件的简化版本，原来的文件大约有 500 行

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<feed xmlns="http://www.w3.org/2005/Atom"
    xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:flickr="urn:flickr:"xmlns:media="http://search.
    ➝yahoo.com/mrss/">

    <title>Content from Paradise Ridge Sculpture Grove</title>
    <link rel="self" href="http://api.flickr.com/services/feeds/photoset.gne?set=72157600976524175&amp;
    ➝nsid=23922109@N00&amp;lang=en-us" />
```

```
<link rel="alternate" type="text/html" href="http://www.flickr.com/photos/dorismith/sets/
➝72157600976524175"/>
<id>tag:flickr.com,2005:http://www.flickr.com/photos/23922109@N00/sets/72157600976524175</id>
<icon>http://farm2.static.flickr.com/1335/882568164_72eee9b41f_s.jpg</icon>
<subtitle>The &lt;a href=&quot;http://www.paradiseridgewinery.com/&quot;&gt;Paradise Ridge Winery&lt;
➝/a&gt; not only has great wines, but they also have a sculpture garden. We visited on 22 July 2007.
➝</subtitle>
<updated>2007-07-24T05:19:08Z</updated>
<generator uri="http://www.flickr.com/">Flickr</generator>

<entry>
   <title>IMG_0045.JPG</title>
   <link rel="alternate" type="text/html" href="http://www.flickr.com/photos/dorismith/882590644/in/
   ➝set-72157600976524175/"/>
   <id>tag:flickr.com,2005:/photo/882590644/in/set-72157600976524175</id>
   <published>2007-07-24T05:19:08Z</published>
   <updated>2007-07-24T05:19:08Z</updated>
   <dc:date.Taken>2007-07-22T13:42:49-08:00</dc:date.Taken>
   <content type="html">&lt;p&gt;&lt;a href=&quot;http://www.flickr.com/people/dorismith/&quot;&gt;
   ➝Dori Smith&lt;/a&gt; posted a photo:&lt;/p&gt;&lt;p&gt;&lt;a href=&quot;http://www.flickr.com/
   ➝photos/ dorismith/882590644/&quot;title=&quot;IMG_0045.JPG&quot;&gt;&lt;img src=&quot;
   ➝http://farm2.static.flickr.com/1063/882590644_5a4a0d89f3_m.jpg&quot; width=&quot;240&quot;
   ➝height=&quot;180&quot; alt=&quot;IMG_0045.JPG&quot;/&gt;&lt;/a&gt;&lt;/p&gt;

   </content>
   <author>
      <name>Dori Smith</name>
      <uri>http://www.flickr.com/people/dorismith/</uri>
   </author>
   <link rel="enclosure" type="image/jpeg" href="http://farm2.static.flickr.com/1063/
    ➝882590644_5a4a0d89f3_m.jpg" />

   <category term="winery" scheme="http://www.flickr.com/photos/tags/" />
   <category term="sonomacounty" scheme="http://www.flickr.com/photos/tags/" />
   <category term="sculptures" scheme="http://www.flickr.com/photos/tags/" />
   <category term="dorismith" scheme="http://www.flickr.com/photos/tags/" />
   <category term="paradiseridge" scheme="http://www.flickr.com/photos/tags/" />
   <category term="paradiseridgesculptures" scheme="http://www.flickr.com/photos/tags/" />
</entry>
<entry>
   <title>IMG_0032.JPG</title>
   <link rel="alternate" type="text/html" href="http://www.flickr.com/photos/dorismith/882568164/in/
   ➝set-72157600976524175/"/>
   <id>tag:flickr.com,2005:/photo/882568164/in/set-72157600976524175</id>
   <published>2007-07-24T05:15:14Z</published>
   <updated>2007-07-24T05:15:14Z</updated>
   <dc:date.Taken>2007-07-22T13:35:09-08:00</dc:date.Taken>
   <content type="html">&lt;p&gt;&lt;a href=&quot;http://www.flickr.com/people/dorismith/&quot;&gt;
   ➝Dori Smith&lt;/a&gt; posted a photo:&lt;/p&gt;&lt;p&gt;&lt;a href=&quot;http://www.flickr.com/
   ➝photos/ dorismith/882568164/&quot; title=&quot;IMG_0032.JPG&quot;&gt;&lt;img src=&quot;
   ➝http://farm2.static. flickr.com/1335/882568164_72eee9b41f_m.jpg&quot; width=&quot;240&quot;
   ➝height= &quot;180&quot;alt=&quot;IMG_0032.JPG&quot;/&gt;&lt;/a&gt;&lt;/p&gt;

   </content>
   <author>
      <name>Dori Smith</name>
      <uri>http://www.flickr.com/people/dorismith/</uri>
   </author>
   <link rel="enclosure"type="image/jpeg"href="http://farm2.static.flickr.com/1335/
   ➝882568164_72eee9b41f_m.jpg" />
```

13

```
        <category term="winery" scheme="http://www.flickr.com/photos/tags/" />
        <category term="sonomacounty" scheme="http://www.flickr.com/photos/tags/" />
        <category term="sculptures" scheme="http://www.flickr.com/photos/tags/" />
        <category term="dorismith" scheme="http://www.flickr.com/photos/tags/" />
        <category term="paradiseridge" scheme="http://www.flickr.com/photos/tags/" />
        <category term="paradiseridgesculptures" scheme="http://www.flickr.com/photos/tags/" />
    </entry>

</feed>
```

⇒ 解析来自服务器的信息

1. `xhr.addEventListener("readystatechange", showPictures, false);`
   `xhr.open("GET", "flickrfeed.xml", true);`

每当 readyState 发生变化时，我们希望调用 showPictures()函数。我们要从服务器读取的文件名是 flickrfeed.xml。这些都在这里进行设置。

2. `var tempText = document.createElement("div");`
   `var theText`

showPictures()中的这些代码执行实际工作。我们首先创建变量来存储两个元素：tempText 和 theText（都是临时的占位符 div）。

3. `var allImages = xhr.responseXML.getElementsByTagName("content");`

服务器发送回的响应包含 XML，所以可以获取响应并寻找每个内容节点。如果看一下脚本 13-8 中的 XML，会看到其中有许多我们根本不感兴趣的东西。实际上，我们需要的只是<a>标签中的内容（其实只需要其中的一半）。所以，我们首先缩小要处理的内容范围。

4. `for (var i=0; i<allImages.length; i++) {`

现在，需要对找到的所有节点执行循环，找出我们需要的实际数据。

5. `tempText.innerHTML = getPixVal(allImages[i]);`

调用 getPixVal()（后面详述）获得第 i 个<content>节点的内容。

因为我们已经得到了 XML 数据，所以可以使用它的 textContent 属性（也可能是 text 属性）获得节点的文本。我们希望找到其中的所有段落——应该有两个段落。

6. `theText= tempText.getElementsByTagName("p")[1].innerHTML;`
   `theText = theText.replace(/240/g,"75");`
   `theText = theText.replace(/180/g,"75");`
   `theText = theText.replace(/_m/g,"_s");`

正如前面提到的，我们只需要一半<a>节点，所以要去掉不需要的那些。在一个包含 20 张照片信息的文件中，就有 20 个<content>节点，每个节点包含两个段落。每个<content>节点包含摄影师的姓名（链接到他们的 Flickr 页面），然后是一个图像，它链接到 Flickr 保存的版本。我们只需要后者，所以只获取 theText 第二个段落中的 innerHTML，也就是段落中的<a>（以及其中包含的<img>标签）。

接下来，使用正则表达式修改结果。Flickr 发送给我们的是中等大小图像的标签，但是我们只需要缩略版本。因为我们的图像要么是 240×180（水平的），要么是 180×240（垂直的），缩略图总是 75×75，所以只需把文本中的所有 240 或 180 改为 75。最后，修改图像名本身。Flickr 中等大小图像的名称以_m 结尾，小图像以_s 结尾，所以要把_m 替换为_s。

7. `document.getElementById("pictureBar").innerHTML += theText;`

在这个循环中，把修改后的节点附加到 HTML 页面的 pictureBar 中。最终结果见图 13-3，这个页面上的每个缩略图都链接到全尺寸版本。

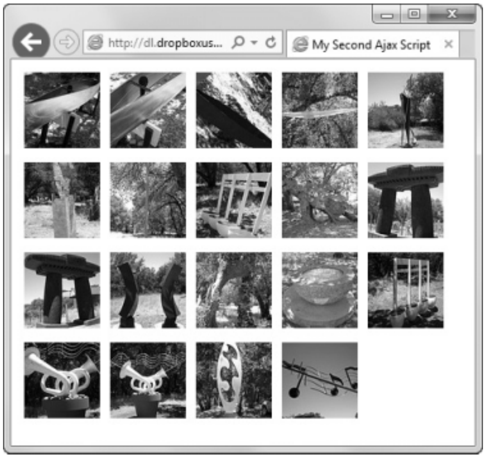


图 13-3　这些缩略图是从 Flickr 读取的

8. `return (inVal.textContent) ? inVal.textContent : inVal.text;`

既然我们已经有了 XML 数据，就可以使用它的 textContent（视具体情况也可能是 text）属性得到节点的文本。这是 getPixVal()函数的全部内容，基本上与脚本 13-2 的 getText()一样，只不过这里用了另一种语法。参见第 2 章补充内容“不存在唯一正确的方式”。

✔提示

- 尽管不能读取存储在另一个服务器上的数据文件（关于这种情况的原因，请参见后面的补充内容“获得数据”），但是可以让 HTML 文件从另一个服务器加载信息。网页无论在什么地方，都能够显示来自 Flickr 服务器的图像。
- Web 网站的优点之一是，它们理解人们希望访问数据——不只是他们自己的数据，而且包括别人的数据（如果数据的主人同意将数据公开的话）。例如，可以在 Flickr 上搜索所有包含夏威夷和日落的照片，然后以 XML 文件的形式返回结果。将此功能与这个脚本或下一个脚本结合在一起，就可以在页面上显示最新的照片。

13

获得数据

当人们第一次听说 Ajax 时，最想做的事情之一是，用 JavaScript 读取各种 XML 文件（包括 RSS 和 Atom 提要），重新组合它们，然后将结果放在自己的网页上。

不好的地方是：这种方式有一点儿限制——脚本只能读取它所在的服务器上的文件。如果你稍微考虑一下，就会明白这是为什么：如果脚本能够读取任何文件，那么就会出现各种安全隐患和假冒的站点。

好的一面在于：可以让服务器上的程序定期运行，获得一个 XML 文件，然后将它存储在本地。

在此之后，Ajax 应用程序就可以读取它。在这个示例和下一个示例中，假设已经运行了某个程序，它定期获取所选择的 Flickr 数据文件，并且将它保存在你自己的服务器上。但是，具体做法超出了本书的范围。

更棒的是，在某些情况下可以使用目标服务器本身驻留的脚本，脚本可以读取自己服务器上的文件，然后把结果返回给用户。我们稍后就会看到一个示例。

13.4 刷新服务器数据

我们的 Ajax 应用程序已经从服务器获得了信息，然后解析数据并且对数据进行操作。现在，我们要演示如何让应用程序从服务器获得数据的新版本，然后自动地刷新页面。脚本 13-9 包含必要的 JavaScript 代码。

脚本 13-9 使用这个脚本自动地刷新服务器信息

```
window.addEventListener("load",initAll,false);
var xhr = false;

function initAll() {
     if (window.XMLHttpRequest) {
        xhr = new XMLHttpRequest();
     }
     else {
        if (window.ActiveXObject) {
           try {
              xhr = new ActiveXObject("Microsoft.XMLHTTP");
           }
           catch (e) {
           }
        }
     }

     if (xhr) {
        getPix();
     }
     else {
        alert("Sorry, but I couldn't create an XMLHttpRequest");
     }
}

function getPix() {
     xhr.open("GET", "flickrfeed.xml", true);
     xhr.addEventListener("readystatechange",showPictures, false);
     xhr.send(null);

     setTimeout(getPix, 5 * 1000);
}

function showPictures() {
     var tempText = document.createElement("div");

     if (xhr.readyState == 4) {
        if (xhr.status == 200) {
           var allImages = xhr.responseXML.getElementsByTagName("content");
           var randomImg = Math.floor(Math.random() * allImages.length);
```

```
            tempText.innerHTML = getPixVal(allImages[randomImg]);
            var thisImg = tempText.getElementsByTagName("p")[1];
            document.getElementById("pictureBar").innerHTML = thisImg.innerHTML;
         }
         else {
            alert("There was a problem with the request " + xhr.status);
         }
      }

      function getPixVal(inVal) {
         return (inVal.textContent) ? inVal.textContent : inVal.text;
      }
}
```

⇒ 刷新服务器信息

1. function getPix() {

```
    xhr.open("GET", "flickrfeed.xml", true);
    xhr.addEventListener("readystatechange", showPictures, false);
    xhr.send(null);

    setTimeout(getPix,5 * 1000);
  }
```

前面的脚本在 initAll()中进行 xhr 调用，而这个脚本将它放在单独的函数 getPix()中。而且，增加了 setTimeout()调用。在脚本获得一个随机图像之后 5 秒，它会再次获取图像。

2. var randomImg = Math.floor (Math.random() * allImages.length);

```
  tempText.innerHTML = getPixVal(allImages[randomImg]);
  var thisImg = tempText.getElementsByTagName("p")[1];
```

与前一个示例中进行的循环不同，这次我们只想要一个随机图像。首先，像 4.10 节中那样，使用 Math.random()和 Math.floor()计算出一个零到图像数量减 1 的随机数。之后，使用这个随机数作为 allImages 数组的索引，获得 getPixVal()函数的精确节点。

3. document.getElementById("pictureBar").innerHTML = thisImg.innerHTML;

现在我们获得一个图像，然后将这个图像放进网页（见图 13-4）。

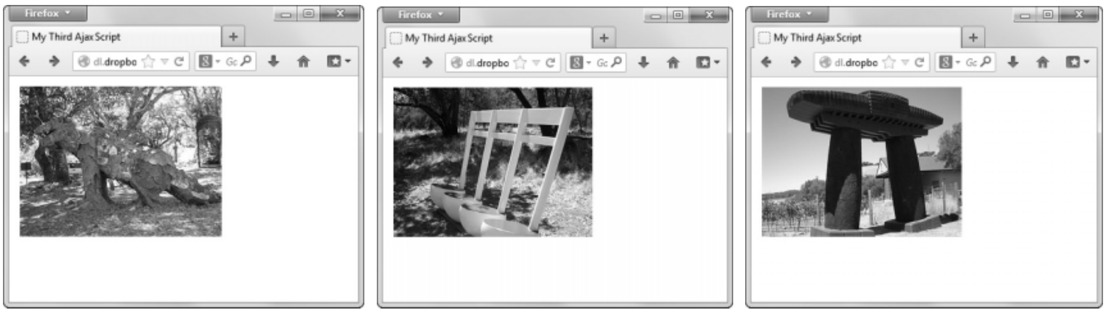


图 13-4　这个脚本一幅接一幅地显示图像

13

✔提示

- 你可能会觉得奇怪：这个脚本为什么每次都读取同一个 XML 文件。如果这个文件没有改动，那么为什么不直接使用第一次获得的变量中的数据？如果你还记得 13.3 节补充内容“获得数据”中提到的技术，就会意识到 XML 文件随时都可能发生改动。假设你的服务器端程序每几分钟创建一次 XML 文件的新版本，为什么用户需要等待才能看到最新的照片呢？这样，用户就能及时地看到它们。
- 如果采用刚才提到的方式，就很可能遇到本章前面提到的 Ajax 缺点：缓存。不同的浏览器（以及不同的版本和平台）都有自己独特的缓存机制，通过修改前面讨论的请求首部，可以解决大多数浏览器（不同版本、不同平台）的缓存机制。许多人推荐的另一个解决方案是将 GET 改为 POST。我们发现另一个方法也是有效的：不同于脚本 13-2，我们在脚本 13-9 中兑换了 open() 和 addEventListener 的次序。

13.5　从服务器获得数据

正如前面补充内容“获得数据”中提到的，Ajax 可以限制读取数据的位置。毕竟，你不希望世界上的每个人都能够读取你的任何文件，不是吗？但是有时候，公司希望别人读取文件，这样就能够在他们自己的网站上创建自己的内容。例如，Flickr（见前面的示例）允许你的服务器获得他们的 XML 文件，然后你可以用这些文件做任何事情。

但是有时候，你无法获得对服务器的访问权，所以 Flickr 以另一种格式提供这些文件：JSON（读音类似人名 Jason）。这个示例的关键之处在 HTML 文件中（见脚本 13-10），JavaScript 文件（见脚本 13-11）只是使用它。

脚本 13-10　这个示例的关键之处是在 HTML 页面中添加远程脚本标签

```
<!DOCTYPE html>
<html>
<head>
     <title>Using JSON Data</title>
     <link rel="stylesheet"href="script02.css">
     <script src="script04.js"></script>
     <script src="http://api.flickr.com/services/feeds/photoset.gne?nsid=23922109@N00&set=
     →72157600976524175&format=json"></script>
</head>
<body>
     <div id="pictureBar"> </div>
</body>
</html>
```

脚本 13-11　JavaScript 文件缩短了，因为大多数工作由远程服务器完成

```
window.addEventListener("load",initAll,false);
var imgDiv = "";

function initAll() {
     document.getElementById("pictureBar").innerHTML = imgDiv;
}

function jsonFlickrFeed(flickrData) {
```

```
    for (var i=0; i<flickrData.items.length;i++) {
      imgDiv += "<img src='";
      imgDiv += flickrData.items[i].media.m.replace(/_m/g,"_s");
      imgDiv += "' alt='" + flickrData.items[i].title + "'>";
    }
}
```

⇛ 读取和解析服务器数据

1. `<script src="http://api.flickr.com/services/feeds/photoset.gne?nsid=23922109@N00&set=` `→72157600976524175&format=json"></script>`

我们在前面说过，脚本只能读取它所在的服务器上的文件。这个规则仍然有效，但是这并不意味着不能调用另一个服务器上的脚本文件。在这个示例中，这个脚本放在 api.flickr.com 服务器上，因此它可以读取这个服务器上的数据。

2. `document.getElementById("pictureBar").innerHTML = imgDiv;`

在 JavaScript 文件中，这行代码在加载时把所有图像放到页面上，见图 13-5。



图 13-5　不但能够从 Flickr 的服务器获得图像，还能够获得创建页面所用的数据

13

3. `function jsonFlickrFeed(flickrData) {`

现在，你肯定想知道设置图像的代码在哪儿，这是另一个关键之处：这些代码主要在数据文件（见脚本 13-12）中。数据文件包含 JavaScript 能够理解的代码，采用 JSON 格式。在这个示例中，数据文件假设要寻找一个名为 `jsonFlickrFeed()`的函数，所以我们在这里创建这个函数。通过参数传递的是存储数据本身的位置。

4.
```
for (var i=0; i<flickrData.items.length; i++) {
  imgDiv += "<img src='";
  imgDiv += flickrData.items[i].media.m.replace(/_m/g,"_s");
  imgDiv += "' alt='" + flickrData.items[i].title + "'>";
}
```

因为数据已经采用 JavaScript 能够理解的格式，所以不需要再做很多工作。在这里，我们遍历 `items`

数组中的所有图像，构造一个将在屏幕上显示（步骤 2）的文本字符串。`items` 中的每个元素包含关于一个图像的各种信息，但是我们只需要 URL，这一信息存储在 media.m 中。然后，使用正则表达式把中等大小的图像转换为缩略图。

脚本 13-12　JSON 文件的片段。注意，它的大小约是 XML 文件的一半，但是包含相同的数据

```
jsonFlickrFeed({
     "title": "Content from Paradise Ridge Sculpture Grove",
     "link": "http://www.flickr.com/photos/dorismith/sets/72157600976524175",
     "description": "The &lt;a href=&quot;http://www.paradiseridgewinery.com/&quot;&gt;Paradise Ridge
     →Winery&lt;/a&gt; not only has great wines, but they also have a sculpture garden. We visited on 22
     →July 2007.",
     "modified": "2007-07-24T05:19:08Z",
     "generator": "http://www.flickr.com/",
     "items": [
        {
          "title": "IMG_0045.JPG",
          "link": "http://www.flickr.com/photos/dorismith/882590644/in/set-72157600976524175/",
          "media": {"m":"http://farm2.static.flickr.com/1063/882590644_5a4a0d89f3_m.jpg"},
          "date_taken": "2007-07-22T13:42:49-08:00",
          "description": "&lt;p&gt;&lt;a href=&quot;http://www.flickr.com/people/dorismith/&quot;&gt;
          →Dori Smith&lt;/a&gt; posted a photo:&lt;/p&gt; &lt;p&gt;&lt;a href=&quot;http://www.flickr.com/
          →photos/dorismith/882590644/&quot; title=&quot;IMG_0045.JPG&quot;&gt;&lt;img src=&quot;
          →http://farm2.static.flickr.com/1063/882590644_5a4a0d89f3_m.jpg&quot; width=&quot;240&quot;
          →height=&quot;180&quot; alt=&quot;IMG_0045.JPG&quot;/&gt;&lt;/a&gt;&lt;/p&gt; ",
          "published": "2007-07-24T05:19:08Z",
          "author": "nobody@flickr.com (Dori Smith)",
          "author_id": "23922109@N00",
          "tags": "winery sonomacounty sculptures dorismith paradiseridge paradiseridgesculptures"
        },
        {
          "title": "IMG_0032.JPG",
          "link": "http://www.flickr.com/photos/dorismith/882568164/in/set-72157600976524175/",
          "media": {"m":"http://farm2.static.flickr.com/1335/882568164_72eee9b41f_m.jpg"},
          "date_taken": "2007-07-22T13:35:09-08:00",
          "description": "&lt;p&gt;&lt;a href=&quot;http://www.flickr.com/people/dorismith/&quot;&gt;
          →Dori Smith&lt;/a&gt; posted a photo:&lt;/p&gt; &lt;p&gt;&lt;a href=&quot;http://www.flickr.com/
          →photos/dorismith/882568164/&quot; title=&quot;IMG_0032.JPG&quot;&gt;&lt;img src=&quot;
          →http://farm2.static.flickr.com/1335/882568164_72eee9b41f_m.jpg&quot; width=&quot;240&quot;
          →height=&quot;180&quot; alt=&quot;IMG_0032.JPG&quot; /&gt;&lt;/a&gt;&lt;/p&gt; ",
          "published": "2007-07-24T05:15:14Z",
          "author": "nobody@flickr.com (Dori Smith)",
          "author_id": "23922109@N00",
          "tags": "winery sonomacounty sculptures dorismith paradiseridge paradiseridgesculptures"
        }
     ]
})
```

✔提示

- 我们在第 10 章介绍对象字面量时介绍过 JSON。JSON 格式本身是对象字面量的一个子集。如果你不熟悉 JSON，可以回顾那一节。
- 不必总是使用步骤 1 中的 URL。实际上，如果这样做，只会得到与这个页面完全相同的结果。Flickr 允许以许多组合方式使用标签、集和组，这样就会得到个性化的结果。访问 Flickr，找到符合期望的网页，在这个页面上找到提要（feed）地址。然后，只需在 URL 的末尾加上 `&format=json`，就能够得到想要的结果。

- ❑ 第 3 步中函数的另一个名称是回调。只要我们将代码放至正确命名的回调函数中，它就会正常运行，但哪怕是名称稍有差别，都不会正常运行。

13.6　用 Ajax 预览链接

现在的许多站点常常采用一种方便的视觉效果：当用户将鼠标指针移动到链接上时，链接的目标页面上的前几行就会出现在鼠标指针下的浮动窗口中（见图 13-6）。这是一个非常容易创建的 Ajax 应用程序。HTML 见脚本 13-13，CSS 见脚本 13-14，JavaScript 见脚本 13-15。



图 13-6　当用户将鼠标指针移动到链接上时，这个脚本读取服务器上的 HTML 文件，并且在预览窗口中显示文件的前几行

脚本 13-13　预览示例的 HTML 页面

```
<!DOCTYPE html>
<html>
<head>
     <title>Previewing Links</title>
     <link rel="stylesheet" href="script05.css">
     <script src="script05.js"></script>
</head>
<body>
     <h2>A Gentle Introduction to JavaScript</h2>
     <ul>
        <li><a href="jsintro/2000-08.html">August column</a></li>
        <li><a href="jsintro/2000-09.html">September column</a></li>
        <li><a href="jsintro/2000-10.html">October column</a></li>
        <li><a href="jsintro/2000-11.html">November column</a></li>
     </ul>
     <div id="previewWin"> </div>
</body>
</html>
```

脚本 13-14　这个 CSS 设置预览弹出窗口的样式

```
#previewWin {
     background-color: #FF9;
     width: 400px;
     height: 100px;
     font: .8em arial, helvetica, sans-serif;
```

13

```
        padding: 5px;
        position: absolute;
        visibility: hidden;
        top: 10px;
        left: 10px;
        border: 1px #CC0 solid;
        clip: auto;
        overflow: hidden;
}

#previewWin h1, #previewWin h2 {
        font-size: 1.0em;
}
```

脚本 13-15　这个 JavaScript 进行服务器请求并且显示弹出窗口

```
window.addEventListener("load",initAll,false);
var xhr = false;
var xPos, yPos;

function initAll() {
        var allLinks = document.getElementsByTagName("a");

        for (var i=0; i< allLinks.length; i++) {
           allLinks[i].addEventListner("mouseover", getPreview,false);
        }
}

function getPreview(evt) {
        if (evt) {
           var url = evt.target;
        }
        else {
           evt = window.event;
           var url = evt.srcElement;
        }
        xPos = parseInt(evt.clientX);
        yPos = parseInt(evt.clientY);

        if (window.XMLHttpRequest) {
           xhr = new XMLHttpRequest();
        }
        else {
           if (window.ActiveXObject) {
              try {
                 xhr = new ActiveXObject("Microsoft.XMLHTTP");
              }
              catch (e) {
              }
           }
        }

        if (xhr) {
           xhr.addEventListener("readystatechange",showContents,false);
           xhr.open("GET", url, true);
           xhr.send(null);
        }
        else {
           alert("Sorry, but I couldn't create an XMLHttpRequest");
        }
```

```
}

function showContents() {
     var prevWin = document.getElementById("previewWin");

     if (xhr.readyState == 4) {
        if (xhr.status == 200) {
           prevWin.innerHTML = xhr.responseText;
        }
        else {
           prevWin.innerHTML = "There was a problem with the request " + xhr.status;
        }
        prevWin.style.top = yPos+2 + "px";
        prevWin.style.left = xPos+2 + "px";
        prevWin.style.visibility = "visible";

        prevWin.addEventListener("mouseout",function(){prevWin.style.visibility = "hidden";}, flase);
     }
}
```

⇒ 使用 Ajax 预览链接

1.
```
var allLinks = document.getElementsByTagName("a");
for (var i=0; i< allLinks.length;i++) {
  allLinks[i].addEventListener("mouseover",getPreview,false);
}
```

这是 initAll()函数的内容，它遍历页面上的所有链接，并且在每个链接上添加 onmouseover 事件处理程序。这个事件处理程序将（你下面将看到的）读取目标页面并且向（可能的）访问者显示预览。

2.
```
if (evt) {
  var url = evt.target;
}
else {
  evt = window.event;
  var url = evt.srcElement;
}
xPos = parseInt(evt.clientX);
yPos = parseInt(evt.clientY);
```

在 getPreview()中，首先需要查明要读取哪个文件，这就要查看事件的属性。根据访问者使用的浏览器不同，URL 保存在 evt.target 或 window.event.srcElement 中。获得了 URL 之后，就可以获得鼠标的 x 和 y 位置供以后使用。

3.
```
var prevWin = document.getElementById("previewWin");

if (xhr.readyState == 4) {
```

使用 Ajax 读取文件之后，现在进入了 showContents()函数。我们将 previewWin 元素存储在 prevWin 中以备后用。当 xhr.readyState 为 4 时，就该显示预览了。

```
4. if (xhr.status == 200) {
     prevWin.innerHTML = xhr.responseText;
   }
   else {
     prevWin.innerHTML = "There was a problem with the request " + xhr.status;
   }
   prevWin.style.top = yPos+2 + "px";
   prevWin.style.left = xPos+2 +
   "px";
   prevWin.style.visibility = "visible";
   prevWin.addEventListener("mouseout", function(){prevWin.style.visibility="hidden";},flase);
```

如果一切正常，那么 `xhr.status` 为 200，而且我们希望放在 `prevWin.innerHTML` 中的数据已经存在于 `xhr.responseText` 中了。如果出现了问题，就在 `prevWin.innerHTML` 中放一个错误消息。

在此之后，需要查明要在哪里显示预览窗口，也就是当前的鼠标 x 和 y 坐标。这个窗口是一个弹出窗口，所以将它放在触发该调用的当前鼠标位置向下和向右一点儿的地方（向下和向右各 2 像素）。

最后，将 `prevWin` 设置为可见，并且让 JavaScript 知道，当鼠标离开预览窗口时，应该隐藏 `prevWin`。

✔提示

- 读取的数据是 HTML 格式的。将 `xhr.responseText` 放进 `innerHTML` 就会告诉浏览器，当显示预览窗口时，它应该将这里的内容解释为 HTML。如果希望显示别的东西（例如，希望看到页面的实际源代码），那么可以在显示预览之前修改 `innerHTML` 中的内容。
- Ajax 要求被读取的文件驻留在同一服务器上，但是不要求在同一目录中。如果要读取的页面在另一个目录中，而且页面包含相对链接，那么这些链接将不起作用。如果页面引用某个 CSS 文件、图像或 JavaScript，那么不能预览文件的这些部分。对此可以采用同样的解决方案：在显示之前修改 `prevWin.innerHTML`。

13.7 自动补全表单字段

帮助站点访问者的一种非常好的方法是，降低在字段中输入数据的复杂性。帮助用户填写具有大量选项的表单，从而帮助他们节省时间和精力，还有助于向站点提供有效的数据。

例如，当用户在一个表单字段中输入内容时，脚本 13-16（HTML）、脚本 13-17（CSS）和脚本 13-18（JavaScript）会显示与输入的字母匹配的美国州名列表（见图 13-7）。随着用户输入更多的字母，这个列表会逐渐缩短，直到只留下一个州名。然后，这个州名就会自动地放进输入字段中，列表也会消失。

脚本 13-16 这个简单的 HTML 提供将进行自动补全的表单字段

```
<!DOCTYPE html>
<html>
<head>
     <title>Auto-fill Form Fields</title>
     <link rel="stylesheet" href="script06.css">
```

```
    <script src="script06.js"></script>
</head>
<body>
    <form action="#">
       Please enter your state:<br>
       <input type="text" id="searchField" autocomplete="off"><br>
       <div id="popups"> </div>
    </form>
</body>
</html>
```



图 13-7　随着输入更多字母，可能的选择数量逐渐减少

脚本 13-17　这个 CSS 为搜索字段和弹出菜单设置样式

```
body, #searchfield {
    font: 1.2em arial, helvetica, sans-serif;
}

.suggestions {
    background-color: #FFF;
    padding: 2px 6px;
    border: 1px solid #000;
}

.suggestions:hover {
    background-color: #69F;
}

#popups {
    position: absolute;
}

#searchField.error {
    background-color: #FFC;
}
```

脚本 13-18　这个 JavaScript 处理服务器请求和弹出列表的显示

```
window.addEventListener("load",initAll,false);
var xhr = false;
var statesArray = new Array();

function initAll() {
    document.getElementById("searchField").addEventListener("keyup",searchSuggest,false);
```

```
    if (window.XMLHttpRequest) {
       xhr = new XMLHttpRequest();
    }
    else {
       if (window.ActiveXObject) {
          try {
             xhr = new ActiveXObject("Microsoft.XMLHTTP");
          }
          catch (e) {
          }
       }
    }

    if (xhr) {
       xhr.addEventListener("readystatechange",setStatesArray,false);
       xhr.open("GET", "us-states.xml", true);
       xhr.send(null);
    }
    else {
       alert("Sorry, but I couldn't create an XMLHttpRequest");
    }
}

function setStatesArray() {
    if (xhr.readyState == 4) {
       if (xhr.status == 200) {
          if (xhr.responseXML) {
             var allStates = xhr.responseXML.getElementsByTagName("item");
             for (var i=0; i<allStates.length; i++) {
                statesArray[i] = allStates[i].getElementsByTagName("label")[0].firstChild;
             }
          }
       }
       else {
          alert("There was a problem with the request " + xhr.status);
       }
    }
}

function searchSuggest() {
    var str = document.getElementById("searchField").value;
    document.getElementById("searchField").className = "";
    if (str != "") {
       document.getElementById("popups").innerHTML = "";

       for (var i=0; i<statesArray.length;i++) {
          var thisState = statesArray[i].nodeValue;

          if (thisState.toLowerCase().indexOf(str.toLowerCase())== 0) {
             var tempDiv = document.createElement("div");
             tempDiv.innerHTML = thisState;
             tempDiv.addEventListener("click",makeChoice,false);
             tempDiv.className = "suggestions";
             document.getElementById("popups").appendChild(tempDiv);
          }
       }
       var foundCt = document.getElementById("popups").childNodes.length;
       if (foundCt == 0) {
          document.getElementById("searchField").className ="error";
```

```
            }
            if (foundCt == 1) {
               document.getElementById("searchField").value = document.getElementById("popups").firstChild.innerHTML;
               document.getElementById("popups").innerHTML = "";
            }
         }
   }

   function makeChoice(evt) {
         if (evt) {
            var thisDiv = evt.target;
         }
         else {
            var thisDiv = window.event.srcElement;
         }
         document.getElementById("searchField").value = thisDiv.innerHTML;
         document.getElementById("popups").innerHTML = "";
   }
```

⇛ 建立自动补全表单字段

1.
```
Please enter your state:<br>
<input type="text" id="searchField" autocomplete="off"><br>
<div id="popups"> </div>
```

这是我们要注意的 HTML 代码。其中的特殊之处是 autocomplete 属性。它告诉浏览器不要在这个字段上执行任何自动补全，因为我们将用脚本处理自动补全。

2.
```
document.getElementById("searchField").addEventListener("keyup", searchSuggest, false);
```

为了捕捉和处理每次击键，需要一个事件处理程序，这是在 initAll()中设置的。

3.
```
xhr.addEventListener("readystatechange", setStatesArray, false);
xhr.open("GET", "us-states.xml", true);
xhr.send(null);
```

与本章前面的那些照片不同，美国的州名不太可能发生变化。我们可以只读取 XML 文件（见脚本 13-4）一次，对数组进行初始化，并且可靠地假设列表在这次会话结束之前一直是有效的。

4.
```
if (xhr.responseXML) {
    var allStates = xhr.responseXML.getElementsByTagName("item");
    for (var i=0; i<allStates.length; i++) {
      statesArray[i] = allStates[i].getElementsByTagName("label")[0].firstChild;
    }
  }
```

我们在这里读取文件，查看每个 item 节点，寻找其中的 label 节点，并且存储 label 的 firstChild（州名本身）。每个州名存储在 statesArray 数组中的一个元素中。

5.
```
var str = document.getElementById("searchField").value;
document.getElementById("searchField").className = "";
```

当开始在字段中输入内容时，就会执行 searchSuggest()事件处理程序中的代码。首先获得 searchField 的值，也就是到目前为止已经输入的信息。接下来，清空这个字段的 class 属性。

```
6. if (str != "") {
    document.getElementById("popups").innerHTML = "";
```

如果还没有输入任何信息，就不做任何事，所以在这里进行检查，确保用户已经输入了某个值，然后再弹出可能值的列表。如果已经输入了某些信息，就清空以前的可能值列表。

```
7. for (var i=0; i<statesArray.length; i++) {
    var thisState = statesArray[i].nodeValue;
```

现在，遍历州名的列表，并且将当前查看的州名存储在 thisState 中。

```
8. if (thisState.toLowerCase().indexOf(str.toLowerCase()) == 0) {
```

我们希望检查用户到目前为止输入的内容是否是某个州名的一部分——但是仅仅这样还不够，我们还必须确保输入的内容位于州名的开头。毕竟，如果输入了 Kansas，你并不希望下拉框中显示 Arkansas 或 Kansas。另外，在进行这项检查时，还在检查 indexOf()之前确保两个字符串都是小写的。

如果 indexOf()返回 0（也就是说，在 thisState 的开头位置处找到了输入的字符串），那么我们就知道找到了一个匹配。

```
9. var tempDiv = document.createElement("div");
  tempDiv.innerHTML = thisState;
  tempDiv.addEventListener("click",makeChoice,false);
  tempDiv.className = "suggestions";
  document.getElementById("popups").appendChild(tempDiv);
```

因为这个州名是一个可能值，我们希望将它添加到要显示的列表中。实现方法是，创建一个临时的 div，将它的 innerHTML 设置为这个州名，添加 click 处理程序和 className，然后将整个 div 追加到 popups div 中。将每个州名作为单独的 div 添加，这样我们就能够使用 JavaScript 和 CSS 操作每个州名。

```
10. var foundCt = document.getElementById("popups").childNodes.length;
```

当遍历完所有州名之后，我们要建立弹出窗口——但是我们得到了多少个州名呢？这里就计算这个值：foundCt。

```
11. if (foundCt == 0) {
     document.getElementById("searchField").className = "error";
   }
```

如果 foundCt 是 0，就说明用户输入了错误的内容。我们将 className 设置为 error，从而让用户知道输入错了，这一设置会使输入字段显示浅黄色背景（这由脚本 13-17 中的 CSS 样式规则控制）。

```
12. if (foundCt == 1) {
     document.getElementById("searchField").value = document.getElementById("popups").
     →firstChild.innerHTML;
     document.getElementById("popups").innerHTML = "";
   }
```

如果 foundCt 是 1，我们就知道找到了唯一的匹配，所以可以将这个州名放进字段。如果用户已经输入了 ca，他们就不需要再输入 lifornia，因为我们已经知道了他们要输入哪个州名。我们使用 popups 中唯一的 div 填写输入字段，从而自动地提供完整的州名，然后清空 popups div。

```
13. function makeChoice(evt) {
      if (evt) {
        var thisDiv = evt.target;
      }
      else {
        var thisDiv = window.event.srcElement;
      }
      document.getElementById("searchField").value = thisDiv.innerHTML;
      document.getElementById("popups").innerHTML = "";
    }
```

输入州名的另一种方法是，单击弹出列表中的一个州名。在这种情况下，会调用 makeChoice()事件处理程序。首先，我们通过检查事件的目标，查明用户单击了哪个州名，这会提供一个特定的 div。查看这个 div 的 innerHTML 会提供州名，我们将这个州名放进输入字段。最后，清空可能值的弹出列表。

✔提示

- 在使用 Google Instant 时，你会看到这种技术的示例。当你在看似普通的 Google 搜索字段中进行输入时，会显示一个弹出列表，其中显示搜索结果。随着输入的进行，列表中的搜索结果会被不断过滤。
- 你可能会注意到，除了演示 Ajax、XML 和服务器端技术之外，这个示例和前一个示例还花费了许多时间和精力来使效果更美观。这是因为，许多被认为是 Ajax（至少有一部分人这么认为）的东西不但涉及底层技术，而且涉及这些技术的表现方式。接下来几章的目标是使这些技术的应用更加简单。

13.8　检查文件是否存在

到目前为止，本书中已经多次出现图像翻转器示例了。但你有没有想过，如果图像的_on 版本不存在会发生什么？结果如图 13-8 所示。脚本 13-19 所示的代码，在创建翻转器之前使用 Ajax 查看该图像的另一个版本是否存在。



图 13-8　如果图像的翻转器版本不存在，我们会看到一个图像无法显示的图标

脚本 13-19　图像翻转器脚本示例最终版

```
window.addEventListener("load",rolloverInit,false);
var rolloverFound;

function rolloverInit() {
     for (var i=0; i<document.images.length; i++) {
        if (document.images[i].parentNode.tagName.toLowerCase() == "a") {
           setupRollover(document.images[i]);
        }
     }
}

function setupRollover(theImage) {
     var re = /\s*_off\s*/;

     rolloverFound = false;
     if (theImage.src.indexOf("_off")) {
        findImage(theImage.src.replace(re,"_on"));
     }

     if (!rolloverFound) {
        return;
     }
     theImage.outImage = new Image();
     theImage.outImage.src = theImage.src;
     theImage.addEventListener("mouseout",function() {this.src = this.outImage.src;}, false);

     theImage.overImage = new Image();
     theImage.overImage.src = theImage.src.replace(re,"_on");
     theImage.addEventListener("mouseover",function() {this.src = this.overImage.src;}, false);

     theImage.clickImage = new Image();
     theImage.clickImage.src = theImage.src.replace(re,"_click");
     theImage.addEventListener("click",function() {this.src = this.clickImage.src;}, false);

     theImage.parentNode.childImg = theImage;
     theImage.parentNode.addEventListener("blur", function() {this.childImg.src = this.childImg.outImage.
     ➝src;}, false);
     theImage.parentNode.addEventListener("focus", function() {this.childImg.src = this.childImg.overImage.
     ➝src;}, false);
}

function findImage(url) {
     if (window.XMLHttpRequest) {
        xhr = new XMLHttpRequest();
     }
     else {
        if (window.ActiveXObject) {
            try {
                xhr = new ActiveXObject("Microsoft.XMLHTTP");
            }
            catch (e) {
            }
        }
     }

     if (xhr) {
        xhr.addEventListener("readystatechange",picExists,false);
        xhr.open("GET", url, false);
```

```
            xhr.send(null);
        }
    }

    function picExists() {
        if (xhr.readyState == 4) {
            if (xhr.status == 200) {
                rolloverFound = true;
            }
        }
    }
```

⇒ 检查文件是否存在

1. window.addEventListener("road",rolloverInit,false); var rolloverFound;

我们需要一个全局变量，它的作用是一旦发现有效的翻转器_on 图像就进行标记，因此我们在一开始就创建了 rolloverFound 变量。

2. rolloverFound = false;

```
if (theImage.src.indexOf("_off")) {
  findImage(theImage.src.replace(re,"_on"));
}
if (!rolloverFound) {
  return;
}
```

这里展示的是 setupRollover 函数所需的全部改动。开始部分声明 rolloverFound 的默认值为 false。接下来检查图像的 URL 是否包含_off，如果不包含，直接证明了这是个无效的翻转器。如果包含，那么我们还要看一下_on 版本是否存在，检查由后面的 findImage()函数实现。

当我们返回的时候，如果 rolloverFound 依然是 false，立即返回并跳过函数的其他部分。因此不会创建翻转器，这就导致总是显示_off 图像，如图 13-9 所示。

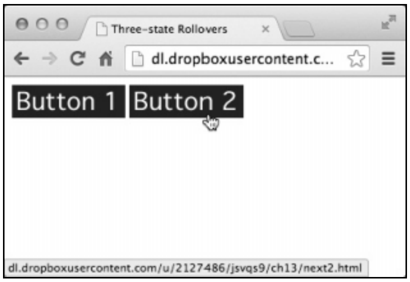


图 13-9　只是通过一个快速的 Ajax 查找，就避免了 UI 问题

3. if (xhr) {

```
    xhr.addEventListener("readystatechange",picExists,false);
    xhr.open("GET", url, false);
    xhr.send(null);
  }
```

findImage()函数中的代码是很标准的 xhr 请求，不过有一点区别：要传递给 xhr.open()的最后一个参数是 false。这就相当于告诉服务器请求是同步的，也就是说，在获得返回结果之前，不能继续。还好，整个检查不会花费太长时间。

4. function picExists() {
```
    if (xhr.readyState == 4) {
      if (xhr.status == 200) {
        rolloverFound = true;
      }
    }
  }
```

最后，检查图片是否真实存在。如果存在，xhr.readysState 是 4，而 xhr.status 是 200——这时就可以将 rolloverFound 设置为 true 了。

第 14 章 工具包、框架和库

有一件事我们之前一直没有提起：编写 JavaScript 应用程序并非易事。一般而言，它要求程序员掌握大量的 DOM、CSS、JavaScript 和服务器资源方面的知识。本书的受众是初级脚本程序员，因此我们介绍了一些简单易懂的知识，以便使读者感觉学习这些知识是力所能及的。不过，许多图书完全面向中高级脚本程序员，介绍如何创建 JavaScript 应用程序，而本书中的章节无法替代那种深入探讨。

那么，这是否意味着如果你尚未成为脚本编程高手，就无法在自己的网站中善用 JavaScript 呢？完全不是！本章将向你展示如何利用 JavaScript 工具包：借助已经预先编写好的库和函数框架，你会在项目实践中真切地感受到脚本编程的强大。

目前有许多工具包可供下载，而且大部分都是免费的。本书选用了可免费下载且开源的实用程序和控件集 jQuery（jquery.com）[①]来构建交互式 Web 应用程序。我们认为它是最出色的 JavaScript 库之一。在接下来的几章中，我们将介绍如何使用 jQuery 来拖放页面元素、添加菜单和日历、创建叠加层和排序表格数据，总的来说，就是利用 jQuery 来为网页添加绚丽实用的效果。

为什么选用 jQuery

无论开发人员有怎样的编程习惯，总会有与之相适应的 JavaScript 框架。换句话说，库的种类千差万别、良莠不齐，而即便是顶尖级的库也各有千秋。

jQuery 库的优点如下。

- **轻量级**：与许多竞争者相比，它的体积要小得多，也就是说，使用 jQuery 的网站加载速度更快。
- **活跃的开发社区**：开发人员有问题可以在论坛上求助，且可以很快得到回复。另外，还可以搜索社区文档资料，查看是否属于常见问题。
- **插件架构**：如果要使用 jQuery 中没有提供的功能，很可能有人已编写好一个插件了。插件的另一个好处是，只有在需要的时候才将其添加到网站中，即不必在每个页面中都加载它。
- **支持旧浏览器**：如果需要支持 IE 6～IE 8，可以使用 jQuery 1 替代通常使用的 jQuery 2，jQuery 1 同样得到了很好的支持。

① 人民邮电出版社出版的《精通 jQuery》（书号 978-7-115-36653-5）是 jQuery 领域的标杆之作，以实例驱动，系统全面讲解了 jQuery、jQuery UI 和 jQuery Mobile：www.ituring.com.cn/book/999。——编者注

- **速度**：即使在由其竞争者所发起的速度测试中，jQuery 亦能胜出（参见 jsperf.com）。
- **初级开发人员容易掌握**：jQuery 的选择查询基于 CSS，因此，非全职专业程序员也能轻松运用 jQuery 来为其网站添加功能，使其按期望的方式工作。

基于上述理由，jQuery 已成为时下最流行的 JavaScript 框架之一。

14.1　添加 jQuery

为了在网站中使用 JavaScript 框架，你需要对网站的页面代码作出一些修改。使用 jQuery 时，代码仅需少量修改。

脚本 14-1　下面的 HTML 代码为页面引入了 jQuery 库

```
<!DOCTYPE html>
<html>
<head>
     <title>Welcome to jQuery!</title>
     <script src="http://code.jquery.com/jquery-2.1.0.js"></script>
     <script src="script01.js"></script>
</head>
<body>
     <h1 id="welcome"> </h1>
</body>
</html>
```

脚本 14-2　一小段 jQuery 代码，这只是个开始

```
$(document).ready(function() {
     alert("Welcome to jQuery!");
});
```

⇒ 在页面中添加 jQuery 库

1. `<script src="http://code.jquery.com/jquery-2.1.0.js"></script>`

脚本 14-1 包含了这句用于引入 jQuery 库的 HTML 代码。

2.
```
$(document).ready(function() {
     alert("Welcome to jQuery!");
});
```

你可能注意到了，脚本 14-2 所示的 JavaScript 文件看起来有些许变化。不必惊慌，上面这段代码和下面这段代码没啥区别：

```
window.addEventListener("load",
  function(){
    alert("Welcome to jQuery!");
  },
  false
);
```

我们在图 14-1 之前已经见过这段代码很多次了，发生变化的地方如下：

- `$()`

在基于 jQuery 的 JavaScript 代码中，人们首先注意到的通常是美元符号$。一方面是由于在普通

JavaScript 代码中$符号并不常见，另一方面则是因为$符号在 jQuery 中无处不在。大多数用到 jQuery 的代码行都会以$符号作为开始，这行代码也不例外。$符号仅仅表示一个合法但不常见的函数名，jQuery 通过它来获取所需的一切。

- ❑ document

因为我们无论做什么都需要通过$，那么，毫无疑问应该首先获取 document 对象（一般是最顶层的元素）。

- ❑ ready()

以前，我们通过检查 onload 事件来判断页面是否完全加载，现在则依赖于 jQuery 的 ready()函数。二者唯一的区别在于 onload 是事件处理程序（所以必须赋给函数），而 ready()是有一个参数的函数。

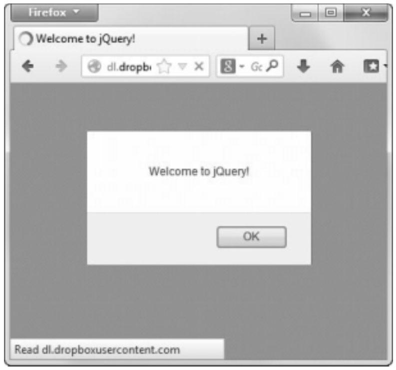


图 14-1 欢迎来到 jQuery 的精彩世界

✔提示

- ❑ 如果你想知道第 1 步中 HTML 代码行的出处，请阅读补充内容“提供 jQuery”。
- ❑ 尽管 jQuery 是最轻量级的 JavaScript 框架之一，还是有办法让它变得更“轻”：下载 jquery-2.1.0.min.js，用它来替换第 1 步中引用的 jQuery 文件。这个版本的代码经过了压缩进而能够获取最快下载速度。

提供 jQuery

以前，如果你想使用 JavaScript 框架（框架的另一个名字是“库”），那么你需要在自己的服务器上提供这些文件。经常访问高端网站的网页冲浪者可能在一天之内下载到各不相同的库中的大部分（或者这些库的多份副本）。

现在对此有一个解决方案：同许多框架一样，从 CDN（Content Delivery Network，内容发布网络）来获取 jQuery。CDN 上能够提供某个库的多个稳定版本，而你可以直接链接到上面。这意味着，如果某个访客正在访问你的网站，而在当天，他曾访问过许多其他使用 jQuery 的网站，那么，他的浏览器端可能已经缓存了所需的库文件，如此一来，你的网站访问速度之快会让人大吃一惊。退一步来说，即使访客的浏览器端没有缓存所需的库文件，CDN 的因特网连接性也要优于你自己实现的网站，所以 CDN 提供文件的速度会优于你的服务器。

对于你而言，仅需对网页做少量修改：通常需要将 HTML 页面中的

```
<script src="directory/script.js">
```

修改为

```
<script src="http://code.jquery.com/script.js">
```

你可能对不断改进的版本感兴趣。写作本书时，jQuery 的最新版本为 2.1.0。如果想获取此版本，可将上面的 URL 改为

```
<script src="http://code.jquery.com/jquery-2.1.0.js">
```

如何获取前一个版本？

```
<script src="http://code.jquery.com/jquery-2.0.3.js">
```

如果要获取 jQuery 1，比方说为了支持 IE 6～IE 8，可以通过加载下面的代码获取 jQuery 1 的最新版。

```
<script src="http://code.jquery.com/jquery.js">
```

（请谨慎使用，这种方法会让开发者的 Web 网站噩梦不断，因此 jQuery 2 不再支持。）

14.2　使用 jQuery 更新页面

到目前为止，本书介绍的都是如何动态更新网页，当然，用 jQuery 也能做到。虽然接下来的示例与前面的示例类似，但我们将会更新页面本身，而不是仅弹出警告框，如图 14-2 所示。



图 14-2　现在，你可以看到由 jQuery 生成的欢迎界面了

脚本 14-3　我们用来修改页面 DOM 结构的 jQuery 代码行

```
$(document).ready(function() {
    $("#welcome").append("Welcome to jQuery!");
});
```

⇒ 使用 jQuery 更新页面

- ❑ `$("#welcome").append("Welcome to jQuery!");`

示例的 HTML 部分与前面类似，不再显示，现在，焦点集中到了脚本 14-3 所示的代码行上。

类似于获取 `document` 对象，使用$从页面上获取元素。现在获取到了`#welcome` 元素。然后，我们使用 jQuery 的 `append()`函数来设置元素的 `innerHTML` 属性。

✓提示

- ❑ 你可能觉得`#welcome` 颇像 CSS，没错！jQuery 流行的原因之一是它的选择器与 CSS 中的选择器非常相似，这样一来，偏重于设计的开发人员就可以更快地掌握它。

14.3　使用 jQuery 交互

前面介绍了一些基础知识，接下来，我们将展示 jQuery 的强大功能。脚本 14-4（HTML）、脚本 14-5（CSS）和脚本 14-6（JavaScript）演示了如何轻松地添加少量用户交互行为。

脚本 14-4　这个 HTML 页面允许用户选择标题的颜色

```
<!DOCTYPE html>
<html>
<head>
     <title>Welcome to jQuery #3!</title>
     <link rel="stylesheet" href="script03.css">
     <script src="http://code.jquery.com/jquery-2.1.0.js"></script>
     <script src="script03.js"></script>
</head>
<body>
     <h1 id="colorMe">Pick a color</h1>
     <p>
        <a id="red">Red</a>
        <a id="green">Green</a>
        <a id="blue">Blue</a>
     </p>
</body>
</html>
```

脚本 14-5　有许多工作要借助 CSS 来完成

```
a {
     display: block;
     float: left;
     padding: 10px;
     margin: 10px;
     font-weight: bold;
     color: white;
     background-color: gray;
}

a:hover {
     color: black;
     background-color: silver;
}
```

页面上有三个像按钮一样的链接，分别是红色、绿色和蓝色。当用户单击其中一个时，页面上的标题“Pick a color”会变成相应选中的颜色，如图 14-3 所示。



图 14-3　单击 Red 按钮会让标题变成红色

脚本 14-6　真正的变化只需寥寥几行 jQuery 代码即可实现

```
$(document).ready(function() {
     $("a").click(function(evt) {
```

14

```
        $("#colorMe").css({
            "color": $(this).attr("id")
        });
        evt.preventDefault();
    });
});
```

⇒ 添加用户交互

1. $("a").click(function(evt) {

此行使用了 jQuery 来实现交互行为，即当用户单击文档上任意链接（即锚点元素）时，会执行相应的操作。

2. $("#colorMe").css({

如你所愿，获取页面上 id 值为 colorMe 的元素。由于我们想改变的属性是 CSS 相关的，因而可以使用 css()函数。它需要一个对象字面量列表：包含在花括号（{}）中的一串名值对，不同的名值对用逗号分隔，名值对的名称和值之间用冒号分隔。

3. "color": $(this).attr("id")

$(this)元素类似于常见的 this 元素，它的值取决于其所处的上下文环境。此时的上下文环境是 click 事件处理程序，因此，我们能够获得用户刚刚单击过的那个链接。

我们真正想要获取的是元素的 id 值，即 red、green 或 blue。将值作为参数传给在第 3 步中仅设置了一个参数（属性名）的 attr()函数。随后，值会被存储为标题的 class 属性的新值，CSS 会接收新值，进而自动更新标题颜色。

其他工具包

在 20 世纪 90 年代，动态 HTML 非常流行，人们编写了大量的 DHTML 工具包。事实上，一部分最优秀的工具包是由无固定职业的自由开发人员在车库里编写的。后来，随着因特网爆炸式地迅速发展，工具包的作者们有了固定的工作，他们遗弃了自己曾经编写过的工具包或者不再对其进行维护。基于此，为本书的早期版本寻找合适的 JavaScript 工具包时，我们颇有点担心找不到。

我们之所以在本书中选择并介绍 jQuery 是因为它的文档齐全、性能优异、开源，而且还有一个庞大的开发者社区积极地为其提供支持，这意味着在本书的生命周期内，它基本上会始终存在。当然，还有很多其他优秀的工具包。甚至还有些网站会对不同的工具包进行打分，此类网站中我们最欣赏维基百科，详见 http://en.wikipedia.org/wiki/Comparison_of_JavaScript_frameworks。

在选择工具包时，最重要的标准莫过于考察它是否在支持 Web 标准方面做得很好。换句话说，工具包要能够跨平台支持所有的主流浏览器。对我们而言，主流浏览器包括 Windows 和 Mac 平台上的 Firefox、Safari、Chrome，以及仅针对 Windows 平台的 IE 9 及更高版本。此外，还有一条标准对于考察工具包优劣很重要，即工具包是否经过全面调试以及它的文档是否齐全。

我们建议你也了解一下下面的这些工具包，它们广为人知、有良好的技术支持、文档齐全且在各自网站上有大量详实的示例可供参考。

- Dojo（dojotoolkit.org）;

- ❑ YUI（developer.yahoo.com/yui/）；
- ❑ Prototype（prototypejs.org）；
- ❑ Modernizr（modernizr.com）；
- ❑ MooTools（mootools.net）。

14.4 交互与更新

如果前面的示例没有让你获得足够多的更新信息，也没有让你感受到很多互动，那么接下来的示例会让你有更深刻的体会。现在，按钮文本也会随之改变颜色，如图 14-4 所示，这样，用户就能知道自己的鼠标停在了哪个按钮上。

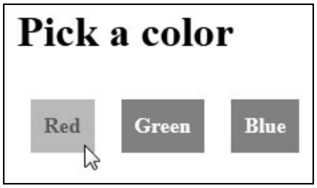


图 14-4 当用户将鼠标停在某个按钮上时，会得到相应的提示

你可能已经猜测到了，HTML 部分与前面的示例相似，不再赘述；脚本 14-7（CSS）和脚本 14-8（JavaScript）包含了所有重要内容。

脚本 14-7 借助 jQuery 完成更多工作后，我们可以对 CSS 代码进行精简

```
a {
    display: block;
    float: left;
    padding: 10px;
    margin: 10px;
    font-weight: bold;
    color: white;
    background-color: gray;
}
```

脚本 14-8 完成这个任务仍然不需要太多的 jQuery 代码

```
$(document).ready(function() {
    $("a").hover(function() {
       $(this).css({
          "color": $(this).attr("id"),
          "background-color": "silver"
       });
    });

    $("a").mouseout(function() {
       $(this).css({
          "color": "white",
          "background-color": "gray"
       });
    });
```

```
      $("a").click(function(evt) {
         $("#colorMe").css({
            "color": $(this).attr("id")
         });
         evt.preventDefault();
      });
});
```

⇒ 进一步增强交互和更新

1. `$("a").hover(function() {`

仅借助 CSS 是无法更新按钮状态的，所以我们需要使用 jQuery 来实现。jQuery 中的 `hover()`函数等价于 `mouseover`。

2. `$(this).css({`

在考虑将鼠标悬停于某个对象上时，我们打算改变它的一些属性。即将被修改的属性均与 CSS 相关，因此我们可以使用 `css()`函数。

3. `"color": $(this).attr("id"),`
   `"background-color": "silver"`

此处，我们会同时修改多个属性，所以需要传入一个对象字面量值列表：一对花括号“{}”中包含的一系列名值对，每个名值对之间用逗号“,”分隔，每个名值对内部的名字和值之间用冒号“:”分隔。将对象字面量列表传给 `css()`函数后，元素的字体颜色和背景颜色会被重设为传入的值。

4. `$("a").mouseout(function() {`

这行代码用于实现 `hover()`的另一半效果：`mouseout()`。用户的鼠标光标移出按钮区域后，我们希望按钮能够恢复到初始的颜色。

5. `"color": "white",`
   `"background-color": "gray"`

这就是我们最初的按钮配色方案——灰底白字。

✓提示

- 如果你想了解更多关于对象字面量的相关知识，可以参考第 10 章的内容。
- 你可能想知道是否真的需要 `mouseout()`部分的代码——毕竟，如果被重置的值是静态的，能否只通过 CSS 来实现呢？很可惜，答案是否定的。`hover()`修改了按钮的颜色后，除非使用 jQuery 重置，否则按钮会一直保持修改后的颜色。

14.5　条纹表格

如果你的网站有大量表格数据，就应该在表格中添加条纹——如果没有条纹，就很难阅读和理解这些信息。遗憾的是，还没有适用于所有常用浏览器的实现条纹表格行的 CSS 方法。在过去，使用 JavaScript 实现条纹表格行非常困难，大多数人不愿意自找麻烦。但是，使用 jQuery 就简单多了，而且效果很好，见图 14-5。



图 14-5　这个专辑列表具有交错的背景，因此很容易阅读

脚本 14-9　这是标准表格的 HTML，没有任何内联样式或内联脚本

```
<!DOCTYPE html>
<html>
<head>
     <title>Striped Tables</title>
     <link rel="stylesheet" href="script05.css">
     <script src="http://code.jquery.com/jquery-2.1.0.js"></script>
     <script src="script05.js"></script>
</head>
<body>
     <h1>Beatles Discography</h1>
     <table>
        <thead>
           <tr>
              <th>Album</th>
              <th>Year</th>
              <th>Label</th>
           </tr>
        </thead>
        <tr>
           <td>Please Please Me</td>
           <td>1963</td>
           <td>Parlophone</td>
        </tr>
        <tr>
           <td>With The Beatles</td>
           <td>1963</td>
           <td>Parlophone</td>
        </tr>
        <tr>
           <td>A Hard Day's Night</td>
```

```
            <td>1964</td>
            <td>Parlophone</td>
         </tr>
         <tr>
            <td>Beatles for Sale</td>
            <td>1964</td>
            <td>Parlophone</td>
         </tr>
         <tr>
            <td>Help!</td>
            <td>1965</td>
            <td>Parlophone</td>
         </tr>
         <tr>
            <td>Rubber Soul</td>
            <td>1965</td>
            <td>Parlophone</td>
         </tr>
         <tr>
            <td>Revolver</td>
            <td>1966</td>
            <td>Parlophone</td>
         </tr>
         <tr>
            <td>Sgt. Pepper's Lonely Hearts Club Band</td>
            <td>1967</td>
            <td>Parlophone</td>
         </tr>
         <tr>
            <td>Magical Mystery Tour</td>
            <td>1967</td>
            <td>Capitol</td>
         </tr>
         <tr>
            <td>The Beatles</td>
            <td>1968</td>
            <td>Apple</td>
         </tr>
         <tr>
            <td>Yellow Submarine</td>
            <td>1969</td>
            <td>Apple</td>
         </tr>
         <tr>
            <td>Abbey Road</td>
            <td>1969</td>
            <td>Apple</td>
         </tr>
         <tr>
            <td>Let It Be</td>
            <td>1970</td>
            <td>Apple</td>
         </tr>
      </table>
</body>
</html>
```

脚本 14-10　将通过 jQuery 启用这些 CSS 代码

```
table {
    border-collapse: collapse;
}

tr.even {
    background-color: #C2C8D4;
}

tr.over {
    background-color: #8797B7;
}

td {
    border-bottom: 1px solid #C2C8D4;
    padding: 5px;
}

th {
    border-right: 2px solid #FFF;
    color: #FFF;
    padding-right: 40px;
    padding-left: 20px;
    background-color: #626975;
}

th.sortUp {
    background: #626975 url(jquery/images/asc.png) no-repeat right center;
}

th.sortDown {
    background: #626975 url(jquery/images/desc.png) no-repeat right center;
}
```

脚本 14-11　这是在表格中添加条纹所需的所有代码

```
$(document).ready(function() {
    $("tr").mouseover(function() {
       $(this).addClass("over");
    });

    $("tr").mouseout(function() {
       $(this).removeClass("over");
    });

    $("tr:even").addClass("even");
});
```

这部分代码起到翻转器的作用：当鼠标移动到某一行上时，触发 `tr` 的 `mouseover`。这让 jQuery 在此行中添加"`over`"类，而 CSS 文件告诉浏览器应该以另一种颜色显示此行（见图 14-6）。



Beatles Discography

Album	Year	Label
Please Please Me	1963	Parlophone
With The Beatles	1963	Parlophone
A Hard Day's Night	1964	Parlophone
Beatles for Sale	1964	Parlophone
Help!	1965	Parlophone
Rubber Soul	1965	Parlophone
Revolver	1966	Parlophone
Sgt. Pepper's Lonely Hearts Club Band	1967	Parlophone
Magical Mystery Tour	1967	Capitol
The Beatles	1968	Apple
Yellow Submarine	1969	Apple
Abbey Road	1969	Apple
Let It Be	1970	Apple

图 14-6　把鼠标悬停在某一行上，就会突出显示该行

⇒ 创建斑马纹表格

1.
```
$("tr").mouseover(function() {
  $(this).addClass("over");
});
```

脚本 14-9（HTML 文件）和脚本 14-10（CSS 文件）没什么特殊之处。唯一有点古怪的地方是，CSS 文件为具有"over"和"even"类的表格行设置了规则，但是在 HTML 中没有设置这些类，因为这项工作是在 JavaScript 文件（见脚本 14-11）中完成的。

2.
```
$("tr").mouseout(function() {
  $(this).removeClass("over");
});
```

这些代码的作用与前面的相反：当鼠标离开该行时，触发它的 `mouseout`，删除类属性"over"。

3.
```
$("tr:even").addClass("even");
```

实际上，这就是添加斑马纹所需的所有代码。因为 jQuery 理解奇数行和偶数行的概念，所以可以让它把所有偶数行的类属性设置为"even"。又因为 CSS 中包含应用于 `tr.even` 的规则，所以会自动设置偶数行的颜色，我们根本不需要修改 HTML。

14.6　表格排序

具有条纹效果的表格已经很不错了，但是有时候你希望站点支持用户交互。用户可能希望按照不同的次序对列进行排序——不是按照年份升序，而是按照降序排序。也可能希望按名称或名称的反序排序。

jQuery 本身并没有提供这种功能，所以必须使用插件。这个插件称为 `tablesorter`。

脚本 14-12　为了给表格添加排序功能，只需在 HTML 中添加少量代码

```
<!DOCTYPE html>
<html>
<head>
```

```
    <title>Sorted Tables</title>
    <link rel="stylesheet" href="script05.css">
    <script src="http://code.jquery.com/jquery-2.1.0.js"></script>
    <script src="jquery/jquery.tablesorter.js"></script>
    <script src="script06.js"></script>
</head>
<body>
    <h1>Beatles Discography</h1>
    <table id="theTable">
       <thead>
          <tr>
             <th>Album</th>
             <th>Year</th>
             <th>Label</th>
          </tr>
       </thead>
       <tr>
          <td>Please Please Me</td>
          <td>1963</td>
          <td>Parlophone</td>
       </tr>
       <tr>
          <td>With The Beatles</td>
          <td>1963</td>
          <td>Parlophone</td>
       </tr>
       <tr>
          <td>A Hard Day's Night</td>
          <td>1964</td>
          <td>Parlophone</td>
       </tr>
       <tr>
          <td>Beatles for Sale</td>
          <td>1964</td>
          <td>Parlophone</td>
       </tr>
       <tr>
          <td>Help!</td>
          <td>1965</td>
          <td>Parlophone</td>
       </tr>
       <tr>
          <td>Rubber Soul</td>
          <td>1965</td>
          <td>Parlophone</td>
       </tr>
       <tr>
         <td>Revolver</td>
          <td>1966</td>
          <td>Parlophone</td>
       </tr>
       <tr>
          <td>Sgt. Pepper's Lonely Hearts Club Band</td>
          <td>1967</td>
          <td>Parlophone</td>
       </tr>
       <tr>
          <td>Magical Mystery Tour</td>
          <td>1967</td>
          <td>Capitol</td>
```

```
        </tr>
        <tr>
            <td>The Beatles</td>
            <td>1968</td>
            <td>Apple</td>
        </tr>
        <tr>
            <td>Yellow Submarine</td>
            <td>1969</td>
            <td>Apple</td>
        </tr>
        <tr>
            <td>Abbey Road</td>
            <td>1969</td>
            <td>Apple</td>
        </tr>
        <tr>
            <td>Let It Be</td>
            <td>1970</td>
            <td>Apple</td>
        </tr>
    </table>
</body>
</html>
```

脚本 14-13　最后，只需添加几行代码，表格就会具备排序功能和条纹效果

```
$(document).ready(function() {
    $("tr").mouseover(function() {
        $(this).addClass("over");
    });

    $("tr").mouseout(function() {
        $(this).removeClass("over");
    });

    $("#theTable").tablesorter({
        sortList:[[1,0]],
        cssAsc: "sortUp",
        cssDesc: "sortDown",
        widgets: ["zebra"]
    });
});
```

⇒ 创建可排序表格

1. `<script src="jquery/jquery.tablesorter.js"></script>`

脚本 14-12（HTML 文件）几乎与脚本 14-9 相同，只有两处改动：加载另一个脚本 jquery.tablesorter.js，在表格中添加 id 值为 theTable 的元素。

2.
```
th.sortUp {
    background: #626975 url(jquery/images/asc.gif)no-repeat right center;
}

th.sortDown {
    background: #626975 url(jquery/images/desc.png)no-repeat right center;
}
```

让我们快速回顾一下脚本 14-10，之前我们跳过了这两条规则。这里，它们可以起作用了。它们告诉浏览器不论用户要升序还是降序，我们都希望表头能显示一个合适朝向的箭头。

3. `$("#theTable").tablesorter({`

这是对 JavaScript 代码（见脚本 14-13）的主要修改：告诉 jQuery，我们希望让用户能够对这个表格的内容排序。这由步骤 3 和步骤 4 实现。我们选择 `id` 为 `theTable` 的元素（见步骤 1）并对它运行 `tablesorter()`方法。

4.
```
sortList:[[1,0]],
cssAsc: "sortUp",
cssDesc: "sortDown",
widgets: ["zebra"]
```

这里使用 jQuery，所以有几个控制表格显示方式的选项。我们在这里使用的选项如下所示。

- ❑ `sortList:[[1,0]]`：在最初加载页面时，我们希望表格以某种方式排序（见图 14-7），就在这里定义这种排序方式。所需排序的列是第一个参数，列的编号从零开始。在这里，我们希望表格按照第二列排序，所以传递 1（请记住，JavaScript 的编号是从零开始的）。第二个参数是希望的排序方向：0 是升序，1 是降序。



Beatles Discography

Album	Year ▼	Label
Please Please Me	1963	Parlophone
With The Beatles	1963	Parlophone
A Hard Day's Night	1964	Parlophone
Beatles for Sale	1964	Parlophone
Help!	1965	Parlophone
Rubber Soul	1965	Parlophone
Revolver	1966	Parlophone
Sgt. Pepper's Lonely Hearts Club Band	1967	Parlophone
Magical Mystery Tour	1967	Capitol
The Beatles	1968	Apple
Yellow Submarine	1969	Apple
Abbey Road	1969	Apple
Let It Be	1970	Apple

图 14-7　在最初加载这个可排序表格时，它按照年份升序排序——Year 标签右边的向下箭头向用户指出了这一点

- ❑ `cssAsc: "sortUp"`：当用户选择升序排序时，我们要对这个 `th` 单元格应用一个新的 CSS 规则。这个选项会在用户选择升序时自动分配`"sortUp"`类，因此会在标签的右边显示向上箭头。
- ❑ `cssDesc: "sortDown"`：如果用户希望采用降序排序，可以再次单击这个 `th` 单元格，这时类会改为`"sortDown"`，因此在标签的右边显示向上箭头（见图 14-8）。因为希望能够按照任何列排序而不只是按照 Web 开发人员决定的列排序，所以用户只需单击另一个 `th` 单元格，结果会立即改变（见图 14-9），不需要添加任何代码。

- ❑ `widgets: ["zebra"]`：在使用 `tablesorter()`时，斑马纹效果是一个“免费赠送的”部件。只需指出希望添加 zebra 部件，就能够产生条纹效果。



Beatles Discography

Album	Year ▲	Label
Let It Be	1970	Apple
Yellow Submarine	1969	Apple
Abbey Road	1969	Apple
The Beatles	1968	Apple
Sgt. Pepper's Lonely Hearts Club Band	1967	Parlophone
Magical Mystery Tour	1967	Capitol
Revolver	1966	Parlophone
Help!	1965	Parlophone
Rubber Soul	1965	Parlophone
A Hard Day's Night	1964	Parlophone
Beatles for Sale	1964	Parlophone
Please Please Me	1963	Parlophone
With The Beatles	1963	Parlophone

图 14-8　单击 Year 标签，表格就会按照降序重新排序，箭头也会变成向上的



Beatles Discography

Album ▼	Year	Label
A Hard Day's Night	1964	Parlophone
Abbey Road	1969	Apple
Beatles for Sale	1964	Parlophone
Help!	1965	Parlophone
Let It Be	1970	Apple
Magical Mystery Tour	1967	Capitol
Please Please Me	1963	Parlophone
Revolver	1966	Parlophone
Rubber Soul	1965	Parlophone
Sgt. Pepper's Lonely Hearts Club Band	1967	Parlophone
The Beatles	1968	Apple
With The Beatles	1963	Parlophone
Yellow Submarine	1969	Apple

图 14-9　单击其他标签（比如 Album），那么此列就会成为排序字段

✔提示

- ❑ 可从 tablesorter.com 下载 `tablesorter` 插件（以及文档和示例）。
- ❑ 使用 jQuery 插件时，代码需要托管在开发者自己的服务器上，这里我们将代码放到了 jquery 目录下。第 16 章会进一步详细讲解插件。

第 15 章 用 jQuery 设计页面

在Web 开发进入需要使用 JavaScript 框架时代的同时，Web 设计也开始频繁考虑使用特定的用户界面元素（比如滑入和滑出屏幕的元素以及无所不在的“黄色渐变”等）。这并非巧合，jQuery 成功的主要原因之一就是它的小伙伴：jQuery UI。闻字识意，jQuery 就是通过 jQuery UI 来处理常用的用户界面元素的。

在本章中，我们将讨论 Web 2.0 界面外观，包括如何创建以及为什么使用它。为了说明这些问题，我们将深入介绍 jQuery 和 jQuery UI 的优点和用法。然后讨论如何突出显示元素、创建可折叠菜单和显示模态对话框，并实现相关联的视觉效果。我们还会涉及更智能的日期选择器，既能显示一个月也能显示两个月。最后为大家展示如何创建自定义主题。

15.1 突出显示新元素

“黄色淡出”差不多已经成了 Web 设计的标志：当页面上出现新内容时，它会先显示在黄色背景上，然后黄色背景慢慢蜕变为白色（即站点的一般背景颜色）。这种巧妙的做法能够提醒访问者注意发生变化的内容，避免了跟踪内容新旧程度的负担。

这个示例还可以很好地说明 jQuery UI，因为只需几行代码，就可以实现丰富的效果。

脚本 15-1 只需在页面开头添加几个<script>标签，就可以添加 jQuery 功能

```
<!DOCTYPE html>
<html>
<head>
     <title>Show/Hide Text</title>
     <script src="http://code.jquery.com/jquery-2.1.0.js"></script>
     <script src="http://code.jquery.com/ui/1.10.4/jquery-ui.js"></script>
     <script src="script01.js"></script>
</head>
<body>
     <a href="#" id="textToggle">show/hidetext</a><br>
     <div id="bodyText">Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Nulla viverra aliquet
     →mi. Cras urna. Curabitur diam. Curabitur eros nibh, condimentum eu, tincidunt at, commodo vitae,
     →nisi. Duis nulla lectus, feugiat et, tincidunt nec,iaculis vehicula, tortor. Sed tortor felis,
     →viverra vitae, posuere et, ullamcorper a, leo. Suspendisse euismod libero at orci. Pellentesque
     →odio massa, condimentum at, pellentesque sed, lacinia quis, mauris. Proin ultricies risus cursus
     →mi. Cras nibh quam, adipiscing vel, tincidunt a, consequat ut, mi. Aenean neque arcu, pretium
     →posuere, tincidunt non, consequat sit amet, enim. Duis fermentum. Donec eu augue. Mauris sit amet
     →ligula.</div>
```

```
</body>
</html>
```

脚本 15-2　这段代码（使用 jQuery 和 jQuery UI）简化了新页面元素的突出显示

```
$(document).ready(function() {
     $("#bodyText").hide();

     $("#textToggle").click(
        function() {
           $("#bodyText").toggle("highlight",{}, 2000);
           return false
        }
     );
});
```

⇒ 突出显示元素

1.
```
<script src="http://code.jquery.com/jquery-2.1.0.js"></script>
<script src="http://code.jquery.com/ui/1.10.4/jquery-ui.js"></script>
<script src="script01.js"></script>
```

为了让脚本 15-1（HTML 页面）能够使用 jQuery UI，它需要访问两个文件：jquery-*version*.js 和 jquery-ui.js。需要的第三个文件是我们自己的本地脚本文件，名为 script01.js。

2. `$(document).ready(function() {`

在脚本 15-2 中，开始编写 JavaScript 例程。这里的代码与在其他地方使用的代码不同，我们传入了一个匿名函数，会在步骤 3 中展示。

3. `$("#bodyText").hide();`

jQuery 最有用的特性之一是指定要操作的对象的方式。这种方式实际上很像 CSS。在编写 CSS 规则时，如果希望隐藏 `id` 为 `bodyText` 的元素，可能会编写下面这样的代码：

```
#bodyText { display:none; }
```

可以看到，CSS 比等效的 JavaScript 命令简短得多：

```
document.getElementById("bodyText").style.display = "none";
```

这一步中代码行的作用与上面的标准 JavaScript 和 CSS 规则的相同：它告诉浏览器不显示指定的元素，见图 15-1。它使用 jQuery 内置的 `hide()`方法，这个方法不需要参数。

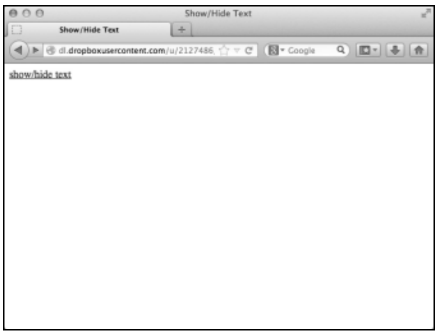


图 15-1　在最初加载页面时，页面上没有太多内容

4. $("#textToggle").click(

在这里，我们调用另一个 jQuery 内置方法：click()。前一步中的代码在加载文档时运行，而这一行由某一事件触发——它在单击 id 为 textToggle 的元素时运行。

需要通过参数向 click()方法传递一个函数，其中包括单击时要触发的代码。

```
5. function() {
     $("#bodyText").toggle("highlight", {}, 2000);
     return false;
   }
```

这是传递给 click()的第一个函数。首先，让 jQuery 寻找 id 为 bodyText 的元素，这是在调用 toggle()时要显示的元素。toggle()方法有 3 个参数。

- "highlight"，我们需要的效果。
- {}，所需的效果选项。黄色淡出技术非常流行，所以黄色是默认颜色，因此这里不需要修改任何选项。
- 2000，希望显示效果的速度。这个值以毫秒为单位，所以 2000 意味着希望在两秒内显示淡出效果。

最后返回 false 值，这样浏览器就不会跟踪链接了。图 15-2 显示最初的淡出效果，图 15-3 显示最终结果。



图 15-2 单击链接，就会在黄色背景上显示文本，然后黄色逐渐淡出

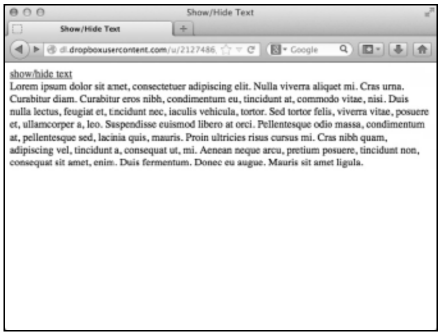


图 15-3 最终的页面显示

✔提示

- 大家是否注意到了 jQuery UI 上面的一行引用？如果没有，那么在此解释一下。这是 jQuery 最好的功能之一：两者可以如此完美地协同工作，我们完全可以不去考虑哪些函数是从谁那儿引入的。在这个例子中，toggle()方法其实是 jQuery UI 的一部分。
- 可能大家会有疑问，为什么 jQuery 不把它的 UI 元素纳入其中？要提醒大家的是，jQuery 使用的很多框架都在后台起作用，没有必要总是把用不到的 UI 效果也一起加载。

15

15.2 创建可折叠菜单

选择框架的一种方法是，确定经常需要在站点中添加的某种功能，然后看看该框架对于完成这个

任务有多大帮助。在这里，我们希望添加可折叠菜单（accordion menu）。在这种菜单中，当打开一个部分时，其他部分会自动关闭。与选项卡式界面相似，它也是一种常用的设计元素。

脚本 15-3　这个大纲中的链接将通过 jQuery 在浏览器中显示为可折叠菜单

```
<!DOCTYPE html>
<html>
<head>
     <title>Accordion Menus</title>
     <link rel="stylesheet" href="http://code.jquery.com/ui/1.10.4/themes/cupertino/jquery-ui.css">
     <link rel="stylesheet" href="script02.css">
     <script src="http://code.jquery.com/jquery-2.1.0.js"></script>
     <script src="http://code.jquery.com/ui/1.10.4/jquery-ui.js"></script>
     <script src="script02.js"></script>
</head>
<body>
     <h1>Shakespeare's Plays</h1>
     <ul id="theMenu">
        <li><a href="menu1.html" class="menuLink">Comedies</a>
           <ul>
              <li><a href="pg1.html">All's Well That Ends Well</a></li>
              <li><a href="pg2.html">As You Like It</a></li>
              <li><a href="pg3.html">Love's Labour's Lost</a></li>
              <li><a href="pg4.html">The Comedy of Errors</a></li>
           </ul>
        </li>
        <li><a href="menu2.html" class="menuLink">Tragedies</a>
           <ul>
              <li><a href="pg5.html">Anthony &amp; Cleopatra</a></li>
              <li><a href="pg6.html">Hamlet</a></li>
              <li><a href="pg7.html">Romeo &amp; Juliet</a></li>
           </ul>
        </li>
        <li><a href="menu3.html" class="menuLink">Histories</a>
           <ul>
              <li><a href="pg8.html">Henry IV, Part 1</a></li>
              <li><a href="pg9.html">Henry IV, Part 2</a></li>
           </ul>
        </li>
     </ul>
</body>
</html>
```

脚本 15-4　尽管我们在前面已经实现了相似的菜单，但是使用 jQuery 可以减少大量 CSS 代码

```
#theMenu {
     width: 400px;
}
```

脚本 15-5　通过使用 jQuery，所需的 JavaScript 减少了

```
$(document).ready(function() {
     $("#theMenu").accordion({
        animated: false,
        collapsible: true,
        header: ".menuLink"
        heightStyle:  "content"
     });
});
```

⇛ 创建可折叠菜单

1.
```
<link rel="stylesheet" href="http://code.jquery.com/ui/1.10.4/themes/cupertino/jquery_ui.css">
<link rel="stylesheet" href="script02.css">
```

脚本 15-3 需要两个 CSS 文件：一个是 Cupertino（jQuery UI 的内置主题），另一个是我们自己的（script02.css，见脚本 15-4），它添加一点儿 CSS 代码，从而实现我们希望的效果。

2.
```
$(document).ready(function() {
```

现在的代码在脚本 15-5 中。与以前一样，如果希望在页面加载时运行某些代码，就需要把代码放在这个函数中。

3.
```
$("#theMenu").accordion({
```

在脚本 15-3 中,通过一个大纲构造菜单,并用无序列表项构造每个菜单的内容。这个示例展示了 jQuery 的简洁性：脚本 15-5 只需获得顶层 `ul` 的 `id`（这里是 `theMenu`），然后对它应用内置的 `accordion()`方法。

4.
```
animated: false,
collapsible: true,
header: ".menuLink"
heightStyle:  "content"
```

我们需要设置几个选项，这在 `accordion()`内部进行。它们如下所示。

- ❑ `animate`：如果希望在显示菜单项时具有动画效果，那么把这个选项设置为所需的效果名称（例如，`"slide"`和`"easeslide"`）。
- ❑ `collapsible`：允许用户折叠所有菜单选项，如图 15-4 所示，这样就可以不必总是显示某些菜单。
- ❑ `header`：jQuery 如何识别每个菜单的标题？在这里，所有菜单标题的类都是`"menuLink"`，单击一个标题，就会显示与图 15-5 相似的页面。
- ❑ `heightStyle`：将它的值设置为`"content"`迫使可折叠区域总是具有固定的高度（基于所需的最大区域）。

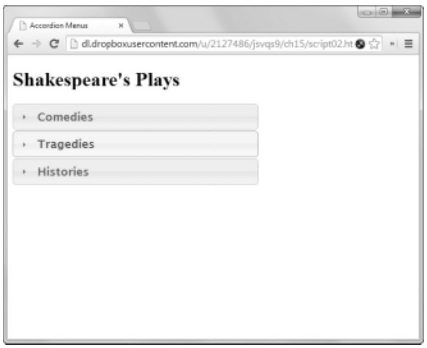


图 15-4 可设置用户能否自由折叠全部菜单

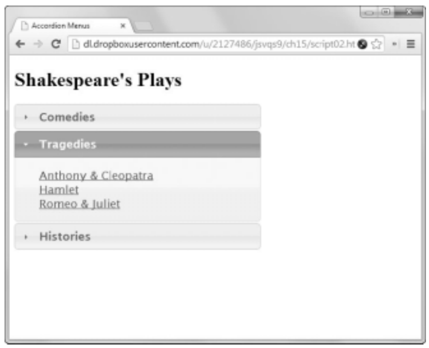


图 15-5 单击一个菜单标题，就会打开一个新的子菜单（并关闭已经打开的其他子菜单）

✔提示

- ❑ 除了上述选项之外，`accordion()`还有其他选项。这里设置的选项只是我们希望覆盖默认设置

的选项。如果希望当鼠标悬停在菜单标签上时打开可折叠菜单（而不是只在用户单击菜单标签时打开），那么只需在最后一步中添加：`event: "mouseover"`.

- 要让折叠菜单在初始状态下是关闭的，在图 15-5 所示的选项列表中添加 `active: false`。如果希望在页面加载时展开折叠菜单，但展开的不是第一个菜单项，那么使用 `active: 2`（或者 3，或者其他可能的情况）。
- 不喜欢 Cupertino 主题，想看看还有没有别的选择？查看补充内容“选择主题，任意主题”。

15.3　创建更漂亮的对话框

现代网站上常用的另一个设计元素是对话框，这些对话框并不像第 2 章讲述的那些普通的 `prompt()`、`alert()`或 `confirm()`对话框。它们更像是在应用程序中看到的对话框，比如模态对话框。对于模态对话框，用户必须作出响应，然后才能返回到网页。同样，在 jQuery 中完成这个任务会更简便。

脚本 15-6　同样，只需在这里添加几个<script>标签，就能够启用 jQuery

```
<!DOCTYPE html>
<html>
<head>
     <title>Modal Dialog</title>
     <link rel="stylesheet" href="http://code.jquery.com/ui/1.10.4/themes/redmond/jquery-ui.css">
     <link rel="stylesheet" href=script03.css>
     <script src="http://code.jquery.com/jquery-2.1.0.js"></script>
     <script src="http://code.jquery.com/ui/1.10.4/jquery-ui.js"></script>
     <script src="script03.js"></script>
</head>
<body>
     <div id="example" title="This is a modal dialog">
        So long as you can see this dialog,you can't touch the page below
     </div>
     <h1>Welcome to my page</h1>
     <div id="bodyText">Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Nulla viverra aliquet
     ➝mi. Cras urna. Curabitur diam. Curabitur eros nibh, condimentum eu, tincidunt at, commodo vitae,
     ➝nisi. Duis nulla lectus, feugiat et, tincidunt nec, iaculis vehicula, tortor. Sed tortor felis,
     ➝viverra vitae, posuere et, ullamcorper a, leo. Suspendisse euismod libero at orci. Pellentesque odio
     ➝massa, condimentum at, pellentesque sed, lacinia quis, mauris. Proin ultricies risus cursus mi.
     ➝Cras nibh quam, adipiscing vel, tincidunt a, consequat ut, mi. Aenean neque arcu, pretium posuere,
     ➝tincidunt non, consequat sit amet, enim. Duis fermentum. Donec eu augue. Mauris sit amet ligula.</div>
</body>
</html>
```

脚本 15-7　这一小段代码就能实现对话框的深色背景

```
.ui-widget-overlay {
    background: #000 none;
}
```

脚本 15-8　只需少量 jQuery 代码即可处理模态对话框

```
$(document).ready(function() {
     $("#example").dialog({
        modal: true,
        resizable: false,
        buttons: [{
           text: "OK"
```

```
            click: function() {
               $(this).dialog("close");
            }
         }]
      });
   });
```

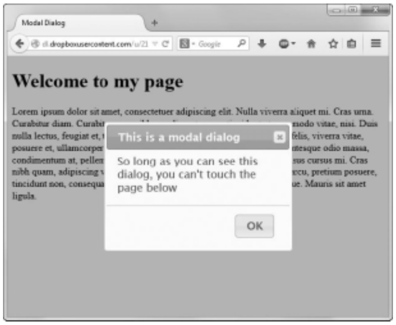


图 15-6　可以在浏览器窗口中拖动这个模态对话框，但是在关闭它之前无法访问它后面的页面

⇒ 创建更漂亮的对话框

1.
```
<link rel="stylesheet" href="http://code.jquery.com/ui/1.10.4/themes/redmond/jquery-ui. css">
<link rel="stylesheet" href=script03.css>
```

在这里，我们使用了 jQuery UI 的另一个内置方案 Redmond。脚本 15-6 中的这行代码把它加载到页面中。另外我们还在特定 CSS 页面加载了它，如脚本 15-7 所示。

2.
```
$("#example").dialog({
   modal: true,
   resizable: false,
```

我们希望在页面上显示一个可拖动的模态对话框，我们需要的定制代码见脚本 15-8。当最初加载页面时运行这些代码，它查找 `example` 元素并用这个元素作为对话框的基础。这个对话框是模态的（因为有 `modal: true`），而且不能调整大小（因为有 `resizable: false`），如图 15-6 所示。

3.
```
buttons: [{
   text: "OK"
   click: function() {
     $(this).dialog("close");
   }
 }]
```

这个对话框只有一个按钮，即 OK 按钮。当单击这个按钮时，对话框关闭。如果需要更多按钮或执行更多操作，应该在这里编写代码。

✔提示

- 默认情况下，对话框是可调整大小和可拖动的。如果需要可调整大小的对话框，只需删除步骤 2 中的 `resizable: false`。

选择主题，任意主题

截至目前为止，读者肯定意识到了本书作者并不是专业的 Web 设计人员。如果恰好你也一样，那肯定会感谢 jQuery UI 免费提供的大量专业主题（参见图 15-7）。

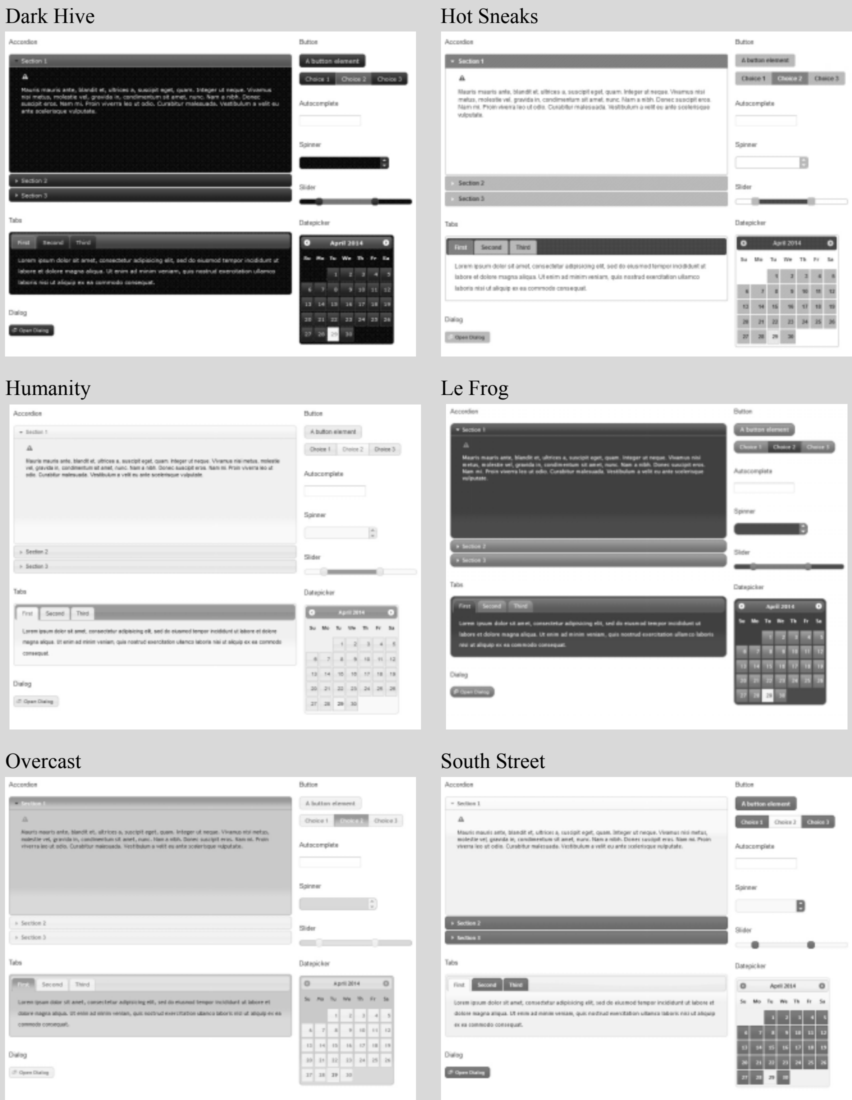


图 15-7　可选的主题有很多：Dark Hive、Hot Sneaks、Humanity、Le Frog、Overcast、South Street、Swanky Purse、Vader。更不用说本章其他地方用过的其他主题了，而且当然，它们看起来颜色更漂亮！

Swanky Purse　　Vader

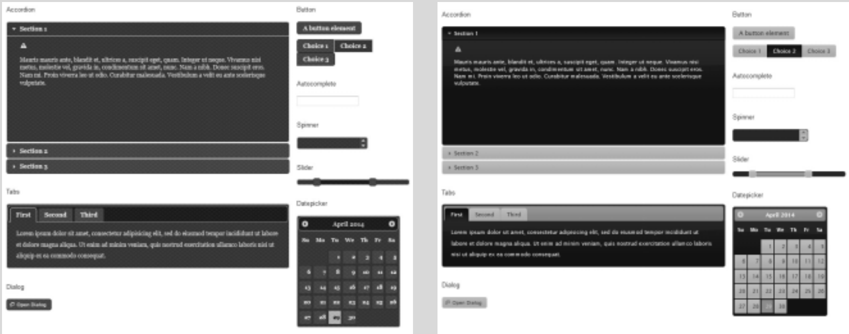


图 15-7　（续）

第 14 章讨论过如何使用 jQuery 的 CDN 来修改 jQuery 的 JavaScript 文件。可喜的是，对于 jQuery UI 的 CSS 文件，也可以这样做。

这次的路径不再是/jquery-version#/，CSS 位于/ui/version#/themes/themeName/jQuery-ui.css，其中：

- version#是我们希望使用的 jQuery UI 的版本号（不是 jQuery 的版本号）；
- themeName 是不断增加的可用主题序号（当前是 24）。在 http://blog.jqueryui.com 上可以找到当前名称和位置列表。

对比脚本 15-3 和脚本 15-6 可以发现，要改变页面风格，唯一需要做的就是编辑<link>标签中的主题名称。这样便于在不同主题之间切换，所以把所有的主题都试一下，看看喜欢哪个吧。

15.4　自动完成字段

你可能记得，在上一章的末尾，我们编写了大量的代码来阻止用户输入无效的美国州名。而在这里，你将看到如何利用 jQuery 来化繁为简。脚本 15-9 是 HTML 代码，脚本 15-10 是 JavaScript 代码。

脚本 15-9　HTML 页面引入了功能实现所依赖的工具包

```
<!DOCTYPE html>
<html>
<head>
     <title>Auto-fill Form Fields</title>
     <link rel="stylesheet" href="http://code.jquery.com/ui/1.10.4/themes/smoothness/jquery-ui.css">
     <script src="http://code.jquery.com/jquery-2.1.0.js"></script>
     <script src="http://code.jquery.com/ui/1.10.4/jquery-ui.js"></script>
     <script src="script04.js"></script>
</head>
<body>
     <div class="ui-widget">
        <label for="searchField">Please enter your state:</label>
        <input id="searchField">
     </div>
</body>
</html>
```

脚本 15-10　jQuery 化繁为简

```
$(function(){
    var stateList = "Alabama*Alaska*Arizona*Arkansas*California*Colorado*Connecticut*Delaware*
    →Florida*Georgia*Hawaii*Idaho*Illinois*Indiana*Iowa*Kansas*Kentucky*Louisiana*Maine*
    →Maryland*Massachusetts*Michigan*Minnesota*Mississippi*Missouri*Montana*Nebraska*Nevada*New
    →Hampshire*New Jersey*New Mexico*New York*North Carolina*North Dakota*Ohio*Oklahoma*Oregon*
    →Pennsylvania*Rhode Island*South Carolina*South Dakota*Tennessee*Texas*Utah*Vermont*Virginia*
    →Washington*West Virginia*Wisconsin*Wyoming*";

    $("#searchField").autocomplete({
       source: stateList.split("*")
    });
});
```

⇛ 使用 jQuery 处理自动完成字段

1. var stateList = "Alabama*Alaska*Arizona*Arkansas*California*Colorado*Connecticut* Delaware*
→Florida*Georgia*Hawaii*Idaho*Illinois*Indiana*Iowa*Kansas*Kentucky*Louisiana*Maine*
→Maryland*Massachusetts*Michigan*Minnesota*Mississippi*Missouri*Montana*Nebraska*Nevada*
→New Hampshire*New Jersey*New Mexico*New York*North Carolina*North Dakota*Ohio*Oklahoma*
→Oregon*Pennsylvania*Rhode Island*South Carolina*South Dakota*Tennessee*Texas*Utah*
→Vermont*Virginia*Washington*West Virginia*Wisconsin*Wyoming*";

我们需要州名列表，而这次，我们将其硬编码到变量 stateList 中。一共有 50 个州名，它们之间以星号分隔。

2. $("#searchField").autocomplete({

每当用户在 searchField 文本框中输入内容时，执行相应操作。

3. source: stateList.split("*")

autocomplete()函数有一个需要赋值的元素——source。source 元素的值类型是数组，那么，我们要做的就是使用 split()函数将 stateList 变量中的值依据星号分隔并形成一个数组。将获得的数组传入后，余下的全部工作交由 jQuery 来完成，如图 15-8 所示。

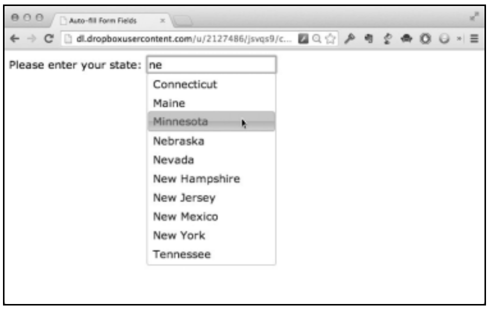


图 15-8　此菜单支持用户通过输入、单击或者方向键上下控制的方式从下拉列表中选择州名

✔提示

- 本例与前面章节中的示例不尽相同。例如，本例中没有限制必须从州名的起始位置进行匹配。另一方面，本例实现了前例中没有的重要功能：它支持用户使用方向键来从下拉菜单中选择州名。

15.5　添加可排序选项卡

到目前为止，你应该对将无序列表转换为菜单、折叠菜单以及其他一些东西习以为常了。在本节的示例中，我们使用链接组成的无序列表展示菜单名称及其所含内容的 div。脚本 15-11 是 HTML 代码，脚本 15-12 是 CSS 代码。脚本 15-13 是必需的最简代码。多亏 jQuery，只需要几行代码就能让菜单项可以调整顺序了。

脚本 15-11　在这个 HTML 页面中，注意链接目标匹配你想查看内容的 id

```
<!DOCTYPE html>
<html>
<head>
     <title>Sortable Tabs</title>
     <link rel="stylesheet" href="//code.jquery.com/ui/1.10.4/themes/pepper-grinder/jquery-ui.css">
     <link rel="stylesheet" href="script05.css">
     <script src="http://code.jquery.com/jquery-2.1.0.js"></script>
     <script src="http://code.jquery.com/ui/1.10.4/jquery-ui.js"></script>
     <script src="script05.js"></script>
</head>
<body>
     <h2>Compact Stars</h2>
     <div id="tabs">
        <ul>
           <li><a href="#tabs-1">Neutron Star</a></li>
           <li><a href="#tabs-2">Pulsar</a></li>
           <li><a href="#tabs-3">Black Hole</a></li>
        </ul>
        <div id="tabs-1">
           <p><span><img src="images/Neutron_star.png" alt="neutron star"><br>Wikimedia Commons</span> A
           →<b>neutron star</b> is a type of stellar remnant that can result from the gravitational
           →collapse of a massive star during some kinds of supernova events. Neutron stars are
           →the densest and tiniest stars known to exist in the universe; although having only
           →the diameter of about 10 km (6 mi), they may have a mass of several times that of the
           →Sun. Neutron stars probably appear white to the naked eye.
           </p>
        </div>
        <div id="tabs-2">
           <p><span><img src="images/pulsar.jpg" alt="pulsar"><br>NASA's Marshall Space Flight
           →Center</span>A <b>pulsar</b> is a highly magnetized, rotating neutron star that emits
           →a beam of electromagnetic radiation. This radiation can only be observed when the beam
           →of emission is pointing toward the Earth, much the way a lighthouse can only be seen
           →when the light is pointed in the direction of an observer, and is responsible for
           →the pulsed appearance of emission.
           </p>
        </div>
        <div id="tabs-3">
           <p><span><img src="images/blackhole.jpg" alt="black hole"><br>NASA's Marshall Space
           →Flight Center</span>A <b>black hole</b> is defined as a region of spacetime from which
           →gravity prevents anything, including light, from escaping. The theory of general
           →relativity predicts that a sufficiently compact mass will deform spacetime to form
           →a black hole. The hole is called "black" because it absorbs all the light that hits
           →its event horizon, reflecting nothing, just like a perfect black body in thermodynamics.
```

```
        ➝The discovery of neutron stars sparked interest in gravitationally collapsed compact
        ➝objects as a possible astrophysical reality. Black holes of stellar mass are expected
        ➝to form when very massive stars collapse at the end of their life cycle. After a black
        ➝hole has formed it can continue to grow by absorbing mass from its surroundings. By
        ➝absorbing other stars and merging with other black holes, supermassive black holes
        ➝of millions of solar masses may form. There is general consensus that supermassive
        ➝black holes exist in the centers of most galaxies.
        </p>
      </div>
    </div>
</body>
</html>
```

脚本 15-12　这个示例中的 CSS 代码的唯一作用就是让图像看起来更好看，如果内容中没有图像，可以省去这段代码

```
span {
    float: right;
    margin-left: 1em;
    font-size: .75em;
    text-align: center;
}

img {
    width: 300px;
}

p::after {
    clear: both;
    content: "";
    display: block;
}
```

脚本 15-13　处理选项卡 UI 和排序功能的 jQuery 代码

```
$(function() {
    var tabs = $("#tabs").tabs();
    tabs.find(".ui-tabs-nav").sortable({
      axis: "x",
      stop: function() {
         tabs.tabs("refresh");
      }
    });
});
```

⇛ 添加可排序选项卡

1. `var tabs = $("#tabs"),tabs();`

我们在这行代码中创建了一个新的变量 `tabs`，它会包含所有的选项卡式菜单项以及其中的内容，这是通过获取 `tabs div` 中的全部内容来实现的。选项卡界面工作所需的全部代码就这些，选项卡界面如图 15-9 和图 15-10 所示。

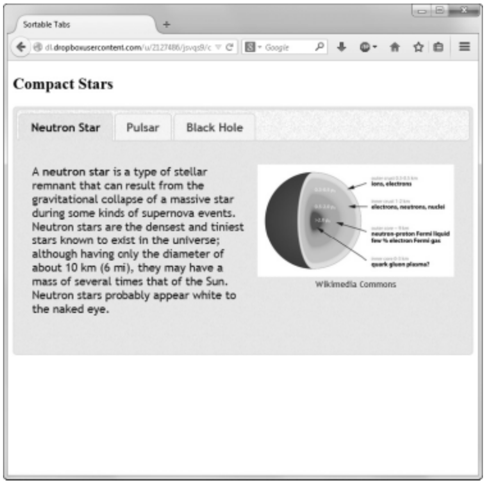


图 15-9 页面加载后自动显示第一个选项卡



图 15-10 单击第二个选项卡会显示第二个选项卡及其所含内容

2.
```
tabs.find(".ui-tabs-nav").sortable({
    axis: "x",
    stop: function() {
        tabs.tabs("refresh");
    }
});
```

这是处理排序的代码。`ui-tabs-nav` 类通过 jQuery 在上一步被应用到了选项卡上，之后我们使用它的 `sortable()`方法定义用户如何重新调整选项卡。此处，我们声明了排序只能沿 x 轴进行，如图 15-11 所示。用户一旦停止拖动选项卡，页面应该马上刷新。



图 15-11 因为菜单是可排序的，用户可以将自己喜欢的选项卡拖到前面

15.6 使用复选框作为按钮

除了无序列表，还有其他一些 HTML 标签也能转换为多姿多彩的 UI 元素。这一节我们就来将单调乏味的复选框变成文雅的按钮，只需一行 jQuery 即可。

脚本 15-14　HTML 设置页面的内容

```
<!DOCTYPE html>
<html>
<head>
     <title>Checkbox Buttons</title>
     <link rel="stylesheet" href="http://code.jquery.com/ui/1.10.4/themes/blitzer/jquery-ui.css">
     <link rel="stylesheet" href="script06.css">
     <script src="http://code.jquery.com/jquery-2.1.0.js"></script>
     <script src="http://code.jquery.com/ui/1.10.4/jquery-ui.js"></script>
     <script src="script06.js"></script>
</head>
<body>
     <div id="stylemenu">
        <input type="checkbox" id="check1"><label for="check1">B</label>
        <input type="checkbox" id="check2"><label for="check2">I</label>
        <input type="checkbox" id="check3"><label for="check3">U</label>
        <input type="checkbox" id="check4"><label for="check4">Code</label>
        <input type="checkbox" id="check5"><label for="check5">ABC</label>
        <input type="checkbox" id="check6"><label for="check6">Abc</label>
     </div>
     <textarea cols="45" rows="10">
     </textarea>
</body>
</html>
```

脚本 15-15　CSS 设置匹配复选框功能的样式

```
#stylemenu {
     padding: 2em 0;
}

#stylemenu input + label {
     font-weight: normal;
}

#stylemenu #check1 + label {
     font-weight: bold;
}

#stylemenu #check2 + label {
     font-style: italic;
}

#stylemenu #check3 + label {
     text-decoration: underline;
}

#stylemenu #check4 + label {
     font-family: monospace;
     font-size: 1.4em;
}

#stylemenu #check5 + label {
     text-decoration: line-through;
}

#stylemenu #check6 + label {
     font-variant: small-caps;
}
```

脚本 15-16 jQuery 非常轻松地就将 HTML 和 CSS 转换为吸引人的 UI 元素

```
$(function() {
     $("#stylemenu").buttonset();
});
```

⇛ 使用复选框作为按钮

1.
```
<div id="stylemenu">
    <input type="checkbox" id="check1">
    <label for="check1">B</label>
    <input type="checkbox" id="check2">
    <label for="check2">I</label>
    <input type="checkbox" id="check3">
    <label for="check3">U</label>
    <input type="checkbox" id="check4">
    <label for="check4">Code</label>
    <input type="checkbox" id="check5">
    <label for="check5">ABC</label>
    <input type="checkbox" id="check6">
    <label for="check6">Abc</label>
</div>
```

这段 HTML 代码（脚本 15-14 的一部分）会正常显示一排复选框及其文本标签。脚本 15-15 中的 CSS 为标签添加样式，匹配其预期功能。

2. `$("#stylemenu").buttonset();`

脚本 15-16 中的这一行 jQuery 代码告诉浏览器找到 `id` 为 `stylemenu` 的元素的内容，并将其转变为标准文本格式化的按钮，如图 15-12 所示。



图 15-12 用户为文本框中的文字选择了粗体和添加删除线的样式

✔提示

- ❑ 注意这个示例中按钮切换开关时，实际上并没有修改输入 `textarea` 字段的文本。

15.7　在页面中添加日历

许多 Web 应用程序都需要一个用户能够参考与交互的日历，如预约表单、待办清单、博客上的博文导航等。jQuery 库包含了一个好用且易于实现的日历控件，如图 15-13 所示。最棒的是它非常灵活，编写寥寥几行代码即可改变它的外观和功能。下面给出一个交互式单日历（一次仅显示一个月）的示例。

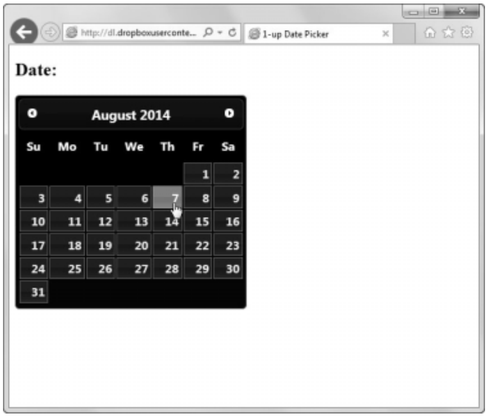


图 15-13　第一次打开页面时，datepicker 部件显示在所选择主题中

脚本 15-17　日历示例的 HTML 页面。请注意，我们指向的是 jQuery 上的 jQuery 副本

```
<!DOCTYPE html>
<html>
<head>
     <title>1-up Date Picker</title>
     <link rel="stylesheet" href="//code.jquery.com/ui/1.10.4/themes/ui-darkness/jquery-ui.css">
     <script src="http://code.jquery.com/jquery-2.1.0.js"></script>
     <script src="http://code.jquery.com/ui/1.10.4/jquery-ui.js"></script>
     <script src="script07.js"></script>
</head>
<body>
     <h2>Date: <span id="datepicked"></span></h2>
     <div id="datepicker"></div>
</body>
</html>
```

脚本 15-18　这个 JavaScript 文件调用了 jQuery，并设置了用户单击日历后显示日期所需的参数

```
$(function() {
     $("#datepicker").datepicker({
        dateFormat: 'DD, MM dd, yy',
        onSelect: function(selectedDate) {
           $("#datepicked").empty().append(selectedDate);
        }
     });
});
```

⇛ 添加单日历

1. `<h2>Date: <span id="datepicked"></span></h2>`
   `<div id="datepicker"></div>`

脚本 15-17 列出了 HTML 页面代码，它需要支持 jQuery 日历控件。

2. `$("#datepicker").datepicker({`

HTML 代码中有一个 id 值为 datepicker 的 div，在脚本 15-18 所示的这行 JavaScript 代码中，我们将 jQuery UI 的 datepicker 部件绑定在了上面。

3. `dateFormat: "DD, MM dd, yy",`

我们准备在页面上以特定的形式显示日期，此处，我们要通知 jQuery 以何种方式显示日期：首先是星期，接着是完整的月份名称，再下来是具体一个月中的哪一天，最后是四位数的年份。

4. `onSelect: function(selectedDate) {`

页面上的日期部件会自动弹出，选中某个日期时，jQuery 事件处理程序 onSelect 会被触发。

5. `$("#datepicked").empty().append(selectedDate);`

选中日期时，我们想要更新页面上的显示，这行代码正是用来做这件事情的。我们会更新 id 值为 datepicked 的 span 标签，首先清空当前值（如果存在的话），然后将 selectedDate 写入其中，如图 15-14 所示。



图 15-14　用鼠标选中了日期以后，被选中的日期会在日历中突出显示。本例中，我们额外添加了在日历上方显示日期的代码

✔提示

- 看起来可能有点奇怪，但是结果正确，yy 会返回四位数的年份。如果要显示两位数的年份，请使用 y 来替换 yy。

⇛ 在页面上添加双联日历

有时，你只需要一个日历，如约会、预订餐厅等。不过，双联日历也很常见。它们通常会被用于那些起止日期不同的事件。例如，预订酒店和购买机票的时候，你常会见到它们。

脚本 15-19　在以下 HTML 页面中添加了双联日历，其中包含两个日期字段

```
<!DOCTYPE html>
<html>
<head>
    <title>2-up Date Picker</title>
    <link rel="stylesheet" href="//code.jquery.com/ui/1.10.4/themes/start/jquery-ui.css">
    Link rel="stylesheet"href="script08.css">
    <script src="http://code.jquery.com/jquery-2.1.0.js"></script>
    <script src="http://code.jquery.com/ui/1.10.4/jquery-ui.js"></script>
    <script src="script08.js"></script>
</head>
<body>
<h1>Select your check in and check out dates:</h1>
    <label for="from">From</label> <input type="text" id="from" name="from">
    <label for="to">to</label> <input type="text" id="to" name="to">
</body>
</html>
```

脚本 15-20　下面的 CSS 代码将 datepicker 下移一小段距离

```
div#ui-datepicker-div {
    margin-top: 10px;
}
```

脚本 15-21　jQuery 函数将 datepicker 对象绑定到了页面的两个日期字段上，并将返回结果存储到一个变量中

```
$(function() {
    var dates = $("#from, #to").datepicker({
       defaultDate: "+1w",
       numberOfMonths: 2,
       onSelect: function(selectedDate) {
          var option = (this.id == "from") ? "minDate" : "maxDate",
          date = $.datepicker.parseDate ($.datepicker._defaults. dateFormat, selectedDate);
          dates.not(this).datepicker ("option", option, date);
       }
    });
});
```

⇛ 添加双联日历

1. `<label for="from">From</label><input type="text" id="from" name="from">`
   `<label for="to">to</label><input type="text" id="to" name="to">`

上述代码是实现功能所需的 HTML 代码的最小集，将其添加到脚本 15-19 所示的 Web 页面中，如图 15-15 所示。脚本 15-20 是 CSS 文件，更短。

2. `var dates = $("#from, #to").datepicker({`

乍一看，脚本 15-21 所示的 JavaScript 代码与脚本 15-18 颇为相似，但实际上并不相同。与之前将 datepicker 对象绑定给页面上的一个元素不同，我们现在将其绑定到了两个元素上：id 值为 from 和 to 的两个元素。此外，我们还将 datepicker 的返回结果保存到了 dates 变量中以备将来使用。

3. `defaultDate: "+1w",`

通过 datepicker 部件，我们可以设置默认的开始日期。这里，我们将其设为下周的今天。

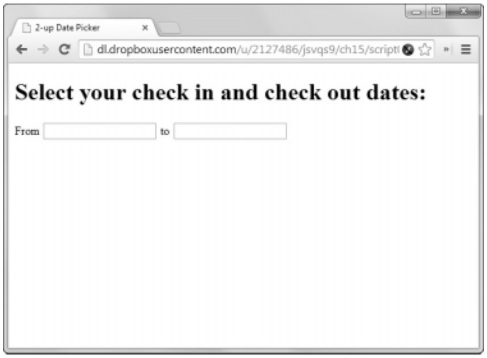


图 15-15 在 datepicker 部件的其他双联形式中，页面会先显示两个日期字段。用 Tab 键进入或者用鼠标单击第一个日期字段时，页面会显示一个双联日历的 datepicker

4. numberOfMonths: 2,

使用 jQuery 的原因之一在于它的灵活性，datepicker 部件灵活性的具体表现之一是它可以轻易地修改每次显示的月份数。此处，我们要一次显示两个月。

5. onSelect: function(selectedDate) {

与之前的示例一样，我们希望在日期被选中的同时做一些事情，具体而言，就是在这个位置编写相应的代码，如图 15-16 所示。



图 15-16 在月份名称栏的左右两侧有箭头形状的按钮，你可以单击它们来更改月份的显示。要选择开始日期，只要单击左侧日历中的日期即可

6. var option = (this.id == "from") ? "minDate" : "maxDate",

此处，我们指出了当前所用的日历，并将结果存储在 option 变量中。如果 this.id 的值为 from，option 变量的值会被设为 minDate。否则，如果 this.id 的值为 to，option 变量的值会被设为 maxDate。

7. date = $.datepicker.parseDate($.datepicker._defaults.dateFormat, selectedDate);

我们能够自动获取到 selectedDate 的值，但其格式并非我们所需的格式。此处，我们使用 datepicker 的 parseDate()函数来转换日期格式，并将结果保存到 date 变量中。

8. dates.not(this).datepicker("option", option, date);

最后，我们用刚刚赋过值的 option 变量和 date 变量来辅助设置可选范围的开始日期（minDate）

和结束日期（maxDate），如图 15-17 所示。



图 15-17 需要注意的是，在左侧的日历中选择了日期以后，选中日期之前的日期都会变灰失效。选中的第一个日期会被填入 datepicker 上方的第一个日期字段中。现在，你可以在右侧的日历中选择第二个日期（高亮显示的部分）了。选中后，其值会被填入到第二个日期字段中

15.8 使用 ThemeRoller 定制外观

作为 Web 开发人员，你需要与设计师一起为网站创建统一的外观。令人高兴的是，jQuery 的作者明白仅为网站添加功能是不够的，还必须考虑这些功能对网站观感的影响。而这正是创建 ThemeRoller 的原因。作为一个工具，ThemeRoller 允许用户根据项目需要来定制 jQuery 的用户界面主题，如图 15-18 所示。你可以新建一个完全自定义的主题，也可以修改任意预先设计好的主题。要了解 ThemeRoller 可以从访问 http://jqueryui.com/themeroller 开始。

⇛ 创建自定义主题

1. 创建自定义主题的最简单的方式莫过于以现有 jQuery 主题中的某一个为基础。单击左侧边栏中的 Gallery，查看你的选择，如图 15-18 所示。

2. 查看可用主题，找到与期望外观最相近的主题。单击主题右下方的 Edit 按钮。随后，左侧边栏会切换至 Roll Your Own 面板，如图 15-19 所示。

3. 此时，你可以选择面板上的可折叠菜单项，单击后会出现相应项的设置，如图 15-20 所示。

4. 编辑侧边栏中的值时，页面的主体部分会进行相应的更新匹配，你可以立即看到（并判断）其差异。

5. 如果你对结果满意，单击面板上方的 Download Theme 按钮，页面会跳转至 Download Builder 页面，如图 15-21 所示。

6. 在这个页面上，你能够控制 CSS 的“轻”与“重”。如果你选择全部组件，那么相比于选择最小集，你的页面会在下载和呈现上花费更长的时间。那就是说，如果你知道自己的网站永远不会用到 Shake 或 Pulsate 效果（晃动或跳动效果），只要不勾选它们前面的复选框，它们就不会被包含在下载范围之内。

7. 选定了所需的全部内容后，单击 Download 按钮。最终，你会获得一个 jQuery 相关的下载文件夹，在其根目录下，有一个名为 index.html 的文件。在浏览器中打开 index.html，它会精确显示出你所

下载的内容，此外，它还会指导你如何将新主题添加到网站页面中。

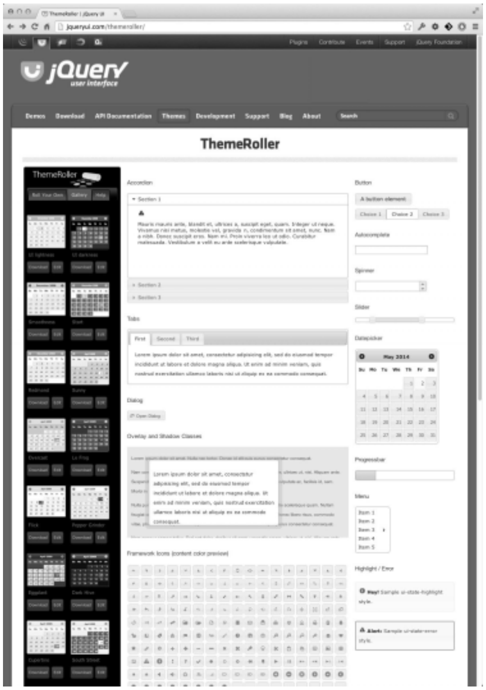


图 15-18 通常情况下，在 ThemeRoller 中定制主题的最佳办法是从 Gallery 选项卡下选出一个现有主题，在其基础上进行自定义

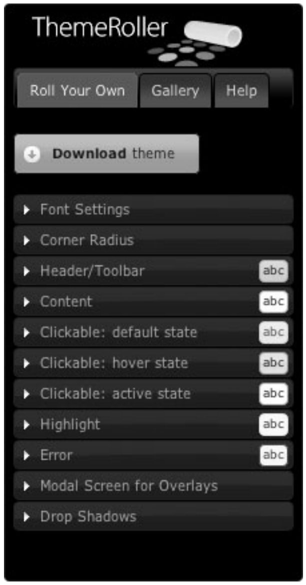


图 15-19 单击 Roll Your Own 选项卡，会显示出更多可用的定制选项

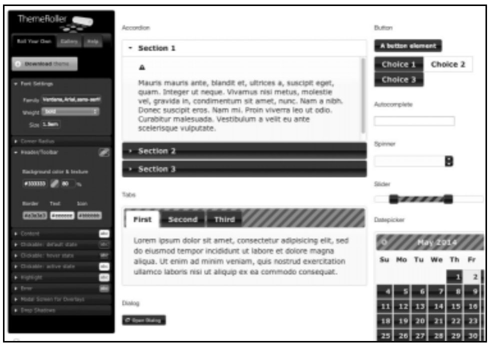


图 15-20 在 Roll Your Own 选项卡中更改类别时，ThemeRoller 会在屏幕右侧为你提供实时预览。此例中，我们将字体粗细程度加强为 Bold，将字号从 1.1em 增加到 1.5em，并设置标题/工具栏（Header/Toolbar）的背景图案为斜纹平行图案。在 Tabs 和 Datepicker 两个组件中，你会看得更加清晰



图 15-21　在 ThemeRoller 中单击 Download Theme 按钮后，页面会跳转至 Download Builder 页面。在这里，你可以进一步定制所需构建和下载的组件。当定制项符合你的需求时，单击 Download 按钮进行下载

✓提示

- 如果使用自定义主题，要确保页面中引用的是刚刚下载的文件而非 jQuery 的 CDN。当然，要记得将它们上传到你的服务器！

第 16 章

基于 jQuery 的应用

在前面两章中，你已经了解了如何利用现有的 HTML 和 CSS 知识，通过 jQuery 库编写尽量少的代码来为网站添加复杂功能。通常情况下，使用 jQuery 在网站中添加 JavaScript 函数比自己编码实现这些函数要容易得多。

上一章中，我们讨论了 jQuery 的用户界面工具包，借助于它，你能够轻松地为网站添加一些功能，如菜单、按钮、对话框和进度条等，这些功能都是 OS X 和 Windows 系统用户的常用功能。除了用户界面上的改进，jQuery 库还为向网站添加各种功能打下了基础。举两个这方面的例子：基于 jQuery，你能够使用 Ajax、JSON 或者两者兼而有之的方式来访问服务器上的远程数据；利用 jQuery 插件，你可以赋予 jQuery 全新的功能，进而为网站添加这些全新的功能。下面，我们来动手实践吧！

16.1　以 jQuery 为基础

引入 jQuery 的最大收获之一是你能将自己从浏览器兼容性的桎梏中解放出来。对不同浏览器而言，其使用 JavaScript 的方式不尽相同，因此，手写 JavaScript 代码时往往需要编写额外的代码以应对不同浏览器的特质。使用 jQuery 时，兼容性问题将不复存在，因为 jQuery 提供了一套跨浏览器的通用函数集。jQuery 库充分考虑了浏览器间的差异，而作为开发者的你则不必再考虑兼容性问题了。

基于 jQuery，使用包括类名和 id 在内的 CSS 选择器，你能够访问并操作页面上的任意元素。你可以更好地控制页面，因为 jQuery 赋予了你随时创建或删除 HTML 元素的能力。

jQuery 库先于页面上其他元素加载（因为它是在页面上的 `head` 部分中调用的，而其他元素则是在 `body` 元素中），因此，待操作的元素一准备好，库就能立即运行与之相关的代码。这种方式优于使用浏览器的 `onload` 函数——只有当页面上包括图片在内的所有元素都加载完毕时，`onload` 函数才会被调用。对用户而言，使用 jQuery 这种方式的好处是页面能够更快响应他们的行为。

16.1.1　Ajax、JSON和jQuery

jQuery 库包含了一系列丰富的用于页面与后台服务器交互的 Ajax 函数。通过这些 Ajax 函数，我们可以获取更多的数据而无需刷新页面。

不刷新页面获取数据的好处在于它让基于 Web 的应用程序可以像桌面应用程序那样快速响应。从服务器接收到了更新过的数据以后，你可以借助 jQuery 来更新页面元素，且不会出现页面刷新时的那种明显闪烁。

正如前面章节中所介绍的，使用 JavaScript 和 `XMLHttpRequest` 对象完全可以自己编写 Ajax 请求。首先，你需要把准备发往服务器的数据加载到对象中，然后，构建另一个函数来接收服务器端的响应。同时，你还需要错误检测代码以确保服务器端的响应是正确而有意义的。如果不想编写上述代码，你仅需使用 jQuery 函数`$.ajax()`即可处理整个过程。

与之类似，jQuery 还提供了`$.getJSON()`函数，让你能够轻松访问和操作接收到的 JSON 格式的数据。

16.1.2　jQuery插件

jQuery 核心库包含了大量的内置功能，尽管如此，开发人员可以轻易地将新功能添加为插件，进而为网站创建一套几乎无限制的功能集。目前，有数以百计的插件可供免费下载，它们在不同的领域扩展了 jQuery 的功能。你能搜到各种功能的插件，比如添加动画效果，允许用户拖放页面元素，改变页面布局，处理不同的媒体类型，操作页面导航，添加有用的部件等。

在你喜爱的 Web 搜索引擎中简单搜索一下，即可找到丰富的可供选择的 jQuery 插件。或者，你也可以先在 http://plugins.jquery.com 中搜索插件。

16.2　拖放元素

UI 功能中最美妙的一个功能莫过于用户能够根据个人喜好来拖放页面元素。本节的示例中，我们将创建一个虚拟的轻量级表格页面，以显示基于 Web 的幻灯片，如图 16-1 所示。你可以在页面上拖放图片，使其按特定的顺序显示，如图 16-2 所示。如果这是一个完整的 Web 应用程序，你甚至可以单击“Build it!”按钮来创建幻灯片，并让其按照选定的顺序进行播放，如图 16-3 所示。脚本 16-1 是页面的 HTML 代码，脚本 16-2 是示例的 CSS 代码，脚本 16-3 是示例的 JavaScript 代码。

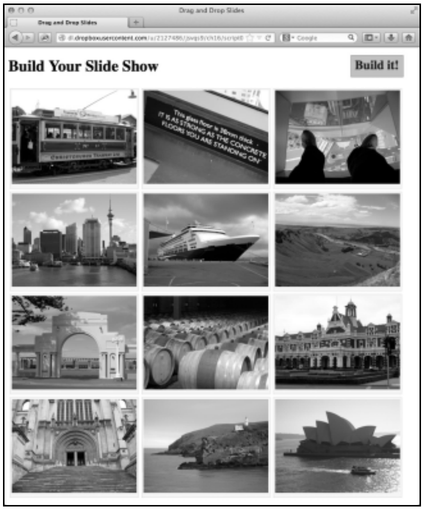


图 16-1　首次加载幻灯片页面时，照片按照预设顺序显示

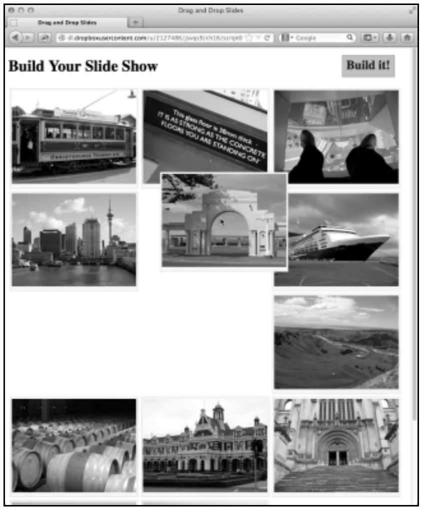


图 16-2　点中并拖动一张图片会使其他图片脱离原来的位置



图 16-3　重新排列后的图片已经可以用于创建幻灯片了，只待你单击“Build it！”按钮（不过，这是另外一个插件了）

脚本 16-1　用于描述虚拟轻量级表格的 HTML 页面中有一个无序列表，其中囊括了所有的图片

```
<!DOCTYPE html>
<html>
<head>
     <title>Drag and Drop Slides</title>
     <link rel="stylesheet" href="//code.jquery.com/ui/1.10.4/themes/flick/jquery-ui.css">
     <link rel="stylesheet" href="script01.css">
     <script src="http://code.jquery.com/jquery-2.1.0.js"></script>
     <script src="http://code.jquery.com/ui/1.10.4/jquery-ui.js"></script>
     <script src="script01.js"></script>
</head>
<body>
     <h1>Build Your Slide Show <a href="#">Build it!</a></h1>
     <ul id="sortable">
        <li class="ui-state-default"><img src="images/img_0001.jpg" alt="slide #1"></li>
        <li class="ui-state-default"><img src="images/img_0002.jpg" alt="slide #2"></li>
        <li class="ui-state-default"><img src="images/img_0003.jpg" alt="slide #3"></li>
        <li class="ui-state-default"><img src="images/img_0004.jpg" alt="slide #4"></li>
        <li class="ui-state-default"><img src="images/img_0005.jpg" alt="slide #5"></li>
        <li class="ui-state-default"><img src="images/img_0006.jpg" alt="slide #6"></li>
        <li class="ui-state-default"><img src="images/img_0007.jpg" alt="slide #7"></li>
        <li class="ui-state-default"><img src="images/img_0008.jpg" alt="slide #8"></li>
        <li class="ui-state-default"><img src="images/img_0009.jpg" alt="slide #9"></li>
        <li class="ui-state-default"><img src="images/img_0010.jpg" alt="slide #10"></li>
        <li class="ui-state-default"><img src="images/img_0011.jpg" alt="slide #11"></li>
        <li class="ui-state-default"><img src="images/img_0012.jpg" alt="slide #12"></li>
     </ul>
</body>
</html>
```

16

脚本 16-2　虚拟轻量级表格的 CSS 文件，其中定义了页面的外观

```
#sortable {
     list-style-type: none;
     margin: 0;
     padding: 0;
     width: 820px;
}

#sortable li {
     margin: 3px;
     padding: 3px;
     float: left;
}

#sortable li img {
     width: 256px;
     height: 192px;
}

h1 a {
     float: right;
     display: inline-block;
     font-size: .8em;
     padding: 8px;
     text-decoration: none;
     background-color: silver;
     border: 1px solid gray;
     margin: -5px 25px 0 0;
}
```

脚本 16-3　只需下面这个简单的 jQuery 函数，用户就能够在页面上拖放图片

```
$(function() {
     $("#sortable").sortable().disableSelection();
});
```

⇒ 启用页面元素的拖放功能

1. <ul id="sortable">

```
<li class="ui-state-default"><img src="images/img_0001.jpg" alt="slide #1"></li>
<li class="ui-state-default"><img src="images/img_0002.jpg" alt="slide #2"></li>
```

上面只列出了 HTML 页面上列表项的前两条（共 12 条）。如果需要增加或减少幻灯片中的图片，只要添加或删除列表中的元素即可。

2. #sortable {

```
list-style-type: none;
margin: 0;
padding: 0;
width: 820px;
}
```

在 CSS 中，待排序的列表项都在 id 值为 sortable 的 div 元素中，而这个 div 包含了页面的主要布局。

3. #sortable li {
    margin: 3px;
    padding: 3px;
    float: left;
  }

每个列表项都需要有一定量的外边距和内边距，同时要被设置成向左浮动，以便与周围其他的列表项紧密相邻。

4. #sortable li img {
    width: 256px;
    height: 192px;
  }

浏览器会按照上面定义的图片尺寸来渲染对应的图片。

5. $("#sortable").sortable().disableSelection();

最后是 jQuery 代码。只有这些？是的，代码就这么多。以上所有的功能只要这么一行 jQuery 代码即可实现。现在，用户可以对 id 值为 sortable 的 div 中的列表项进行排序了，同时，我们屏蔽了选择功能以避免用户在排序过程中选中列表项。

16.3 使用 jQuery 处理外部数据

理论上，在 Web 页面上使用外部数据（XML 或者 JSON 数据）是比较简单的。用户的显示器上有一个页面；服务器上有更多的数据；你希望页面能够在不刷新的前提下加载这些数据。

服务器上的数据可能是文本、图片、音乐、视频等任意形式的数据。下面的示例结合上一节的拖放功能和使用 jQuery 加载第 13 章用过的 Flickr feed。HTML 见脚本 16-4，CSS 见脚本 16-5，JavaScript（随后讲解）见脚本 16-6。

脚本 16-4 获取并显示外部数据 Feed（示例中是 Flickr feed）时，你要先创建 HTML 页面

```
<!DOCTYPE html>
<html>
<head>
     <title>Drag and Drop External Data</title>
     <link rel="stylesheet" href="//code.jquery.com/ui/1.10.4/themes/eggplant/jquery-ui.css">
     <link rel="stylesheet" href="script02.css">
     <script src="http://code.jquery.com/jquery-2.1.0.js"></script>
     <script src="http://code.jquery.com/ui/1.10.4/jquery-ui.js"></script>
     <script src="script02.js"></script>
</head>
<body>
     <h2 id="head"></h2>
     <h4 id="subhead"></h4>
     <ul id="sortable"></ul>
</body>
</html>
```

16

脚本 16-5　Flickr feed 中的元素被解析出来后，CSS 文件就会为它们添加样式

```
#sortable {
     list-style-type: none;
     padding: 0;
}

#sortable li {
     margin: 3px;
     padding: 3px;
     float: left;
}
```

脚本 16-6　这个 JavaScript 文件中的 jQuery 代码用来获取并处理 JSON 格式的 Flickr feed

```
$.getJSON(
     "http://api.flickr.com/services/feeds/photoset.gne?nsid=23922109@N00&set=72157600976524175
     ➝&format=json&jsoncallback=?",
     function(data) {
        createPage(data);
     }
);

function createPage(imgData) {
     var imgs = "";
     $("#head").html(imgData.title);
     $("#subhead").html(imgData.description);

     $.each(imgData.items, function(i, item) {
        imgs += "<li class='ui-state-default'><a href='" + item.link + "'><img src='";
        imgs += item.media.m.replace(/_m/g,"_q") + "' alt='" + item.title + "'></a></li>";
     });

     $("#sortable").append(imgs);
     $("#sortable").sortable().disableSelection();
}
```

⇛ 使用 jQuery 访问 feed 数据

1.
```
$.getJSON(
  "http://api.flickr.com/services/feeds/photoset.gne?nsid=23922109@N00&set=
  ➝72157600976524175&format=json&jsoncallback=?",
  function(data) {
    createPage(data);
  }
);
```

代码乍看起来很复杂，但事实并非如此。传给`$.getJSON`的参数只有两个。

- ❑ 包含所需数据的 URL 字符串。本例中，它是脚本 13-12 中的 Flickr feed。
- ❑ 匿名函数，它在获得数据后会被调用。回调函数只做了一件事：调用了另一个名为`createPage()`的函数。

2.
```
var imgs = "";
$("#head").html(imgData.title);
$("#subhead").html(imgData.description);
```

现在我们进入 createPage()函数，解析输入数据（目前是名为 imgData 的数据数组）以便向页面添加元素。首先初始化新变量 imgs，后面会经常看到这个变量。

接下来获取图像自身的基本信息，并将信息放入 head 和 subhead 元素。

```
3. $.each(imgData.items,function(i, item) {
```

接下来，需要循环处理每个图像。内置函数$.each()会逐行取出数据并存入 item 变量中。

```
4. imgs += "<li class='ui-state-default'><a href='"+ item.link + "'><img src='";
   imgs += item.media.m.replace(/_m/g,"_q") + "' alt='" + item.title + "'></a></li>";
```

关键工作是在这里完成的。对于 feed 中的每个图像，我们为 imgs 变量添加文本，图像变成了图像标签，位于链接标签内部（<a>），而<a>又位于列表元素内部（<li>）。图像链接、图像源以及图像的替换文本（alt）信息均来自 Flickr feed。

```
5. $("#sortable").append(imgs);
   $("#sortable").sortable().disableSelection();
```

完整的 imgs 变量被添加到页面，最终，图像终于又可以调整顺序了，如图 16-4 所示。



图 16-4　jQuery 代码获取到一条 JSON 格式的 feed（本例中，是本书其中一位作者的 Flickr 数据流），解析它，并将结果显示在页面上

16.4　使用 jQuery 插件

如前所述，开发人员为了扩展 jQuery 的核心功能而创建了各式各样的插件。jQuery 插件数量太多，本书不可能一一讨论，因此我们用一个示例来展示 jQuery 插件的强大。

在本例中我们使用插件 SlideJS 显示专业水准的图像幻灯片，如图 16-5 所示。SlideJS 是免费下载软件，跟自定义主题一样，下载后需要集成到自己的网站。为幻灯片添加样式的 CSS 代码会同时下载。



图 16-5　开始播放幻灯片

脚本 16-7　HTML 页面包括来自 SlideJS 插件的文件

```
<!DOCTYPE html>
<html>
<head>
     <title>jQuery Plugin Slideshow</title>
     <link rel="stylesheet" href="SlidesJS/css/example.css">
     <link rel="stylesheet" href="SlidesJS/css/font-awesome.min.css">
     <link rel="stylesheet" href="script03.css">
     <script src="http://code.jquery.com/jquery-2.1.0.js"></script>
     <script src="SlidesJS/source/jquery.slides.min.js"></script>
     <script src="script03.js"></script>
</head>
<body>
     <div id="slides">
        <img src="images/01estock_commonswiki.jpg" alt="01estock_commonswiki.jpg">
        <img src="images/02estock_commonswiki.jpg" alt="02estock_commonswiki.jpg">
        <img src="images/03estock_commonswiki.jpg" alt="03estock_commonswiki.jpg">
        <img src="images/04estock_commonswiki.jpg" alt="04estock_commonswiki.jpg">
        <img src="images/05estock_commonswiki.jpg" alt="05estock_commonswiki.jpg">
        <img src="images/06estock_commonswiki.jpg" alt="06estock_commonswiki.jpg">
        <a href="#" class="slidesjs-navigation slidesjs-previous"><i class="icon-chevron-left icon-large">
        →</i></a>
        <a href="#" class="slidesjs-navigation slidesjs-next"><i class="icon-chevron-right icon-large">
        →</i></a>
    </div>
</body>
</html>
```

脚本 16-8　CSS 文件添加我们所需的一点儿 CSS 样式

```
body {
     padding: 20px;
}

#slides {
     display: none;
}

#slides a {
     color: #333;
}

#slides a:hover,
```

```
#slides a:active {
     color: #9e2020;
}

div.slidesjs-container {
     margin-bottom: 10px;
}

.slidesjs-pagination {
     float: right;
     list-style: none;
     margin: 0;
}

.slidesjs-pagination li {
     float: left;
}

.slidesjs-pagination li a {
     display: block;
     width: 13px;
     height: 0;
     padding-top: 13px;
     background-image: url(SlidesJS/images/pagination.png);
     overflow: hidden;
}

.slidesjs-pagination li a.active,
.slidesjs-pagination li a:hover.active {
     background-position: 0 -13px
}

.slidesjs-pagination li a:hover {
     background-position: 0 -26px
}
```

脚本 16-9 JavaScript 文件中的这几行 jQuery 设置幻灯片的动画效果

```
$(function(){
     $("#slides").slidesjs({
        navigation: {
           active: false,
           effect: "fade"
        },
        pagination: {
           effect: "fade"
        },
        effect: {
           fade: {
              speed: 800
           }
        }
     });
});
```

⇒ 使用 jQuery 插件

1. <link rel="stylesheet" href="SlidesJS/css/example.css">
   <link rel="stylesheet" href="SlidesJS/css/font-awesome.min.css">

插件随附推荐样式表，可以将它们作为一个包使用，像脚本 16-7 这样。同样，之后会加载 jquery.slide.min.js 脚本。脚本 16-8 展示的是本地所需的最少量 CSS 代码。

2. `$("#slides").slidesjs({`

因为引入了 SlideJS 插件的文件，所以我们现在可以使用脚本 16-9 中的 slidejs 函数了。函数设置就在这里完成。

```
3. navigation: {
     active: false,
     effect: "fade"
   },
```

我们想在幻灯片页面中使用自定义导航，因此将 active 设为 false，如图 16-6 所示。我们想让图像出现的时候具有淡入效果，消失的时候具有淡出效果，因此将 effect 设置为"fade"（默认值是"slide"）。

```
4. pagination: {
     effect: "fade"
   },
```

我们也想让不同页面之间切换具有淡入淡出效果，因此将 pagnation 的 effect 也设置为"fade"，如图 16-7 所示。

```
5. effect: {
     fade: {
       speed: 800
     }
   }
```

这段代码使用了淡入淡出效果，我们真是太想显摆这个功能了，设置脚本使用 800 毫秒添加淡入淡出动画效果。



图 16-6　我们可以使用左下角的导航箭头在两张幻灯片之间切换



图 16-7　点击右下角的圆点可以在不同页面之间跳转

✔提示

- SlideJS 的作者是 Nathan Searles。关于插件的信息请查看 slidejs.com。下载网址：github.com/nathansearles/Slides，支持网站：groups.goolge.com/forum/#!forum/slidesjs。
- 跟 jQuery 和 jQuery UI 不同，插件必须先从作者的网站下载，然后上传到自己的服务器，插件没有固定的 CDN。

16.5　添加 jQuery 音频插件

本例中，你会看到如何使用 jQuery 插件来实现一个功能齐全的音频播放器，播放器利用了 HTML5 的 audio 标签，且对于不支持 HTML5 的浏览器使用 Flash 提供向下兼容支持。页面加载时，仅出现几个小按钮，如图 16-8 所示，但是按下播放键后，播放器会展开以显示其所有功能，如图 16-9 所示。

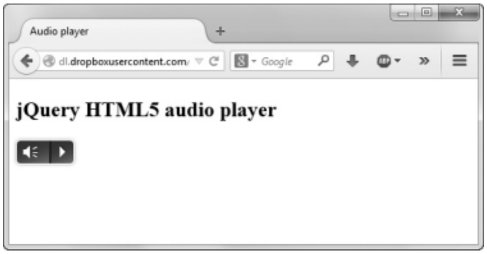


图 16-8　这个 jQuery 插件创建了一个 HTML5 音频播放器。加载后，控制面板会被最小化

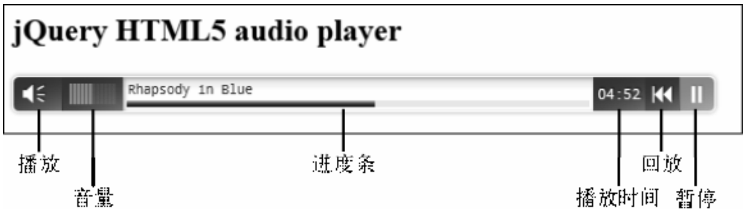


图 16-9　单击播放按钮即开始播放文件，这时，你又可以看到熟悉的播放控制面板了

脚本 16-10　在 HTML 页面上添加 jQuery 插件以提供 HTML5 音频播放器功能

```
<!DOCTYPE html>
<html>
<head>
     <title>Audio player</title>
     <link rel="stylesheet" href="mbplayer/css/miniPlayer.css"/>
     <script src="http://code.jquery.com/jquery-2.1.0.js"></script>
     <script src="mbPlayer/inc/jquery.mb.miniPlayer.js"></script>
     <script src="mbPlayer/inc/jquery.jplayer.min.js"></script>
     <script src="script04.js"></script>
</head>
<body>
     <h2>jQuery HTML5 audio player</h2>
     <div>
     <a class="audio {ogg:'mbPlayer/Rhapsody_in_Blue.ogg'}" href="mbPlayer/Rhapsody_in_Blue.mp3">Rhapsody
     →in Blue</a>
```

16

```
    </div>
</body>
</html>
```

脚本 16-11　通过 jQuery 来调用音频播放器插件，其对应的 JavaScript 文件只有几行代码

```
$(function(){
    $(".audio").mb_miniPlayer({
        width: 360,
        inLine: false,
        showRew: true,
        showTime: true
    });
});
```

⇒ 添加音频播放器

1.
```
<script src="mbPlayer/inc/jquery.mb.miniPlayer.js"></script>
<script src="mbPlayer/inc/jquery.jplayer.min.js"></script>
```

在脚本 16-10 中，脚本标签引入音频插件的两个部分。

2.
```
<a class="audio {ogg:'mbPlayer/Rhapsody_in_Blue.ogg'}" href="mbPlayer/Rhapsody_in_Blue.
mp3">Rhapsody in Blue</a>
```

按照说明（包含在下载文件中）生成链接，加载完成后，播放器会播放我们最喜欢的一首歌曲的 MP3 或者 Ogg 版本。

3.
```
$(".audio").mb_miniPlayer({
    width: 360,
    inLine: false,
    showRew: true,
    showTime: true
});
```

脚本 16-11 包含了我们需要添加的所有 jQuery 代码——其他所需代码都由插件脚本完成了。第 3 步中，我们将播放器的宽度设置为 360 像素，将 `inline` 属性设置成 `false`（即它不属于常规文件流），同时，我们还显示了回放按钮和播放时间。

✓提示

- ❑ 这个插件的作者是 Matteo "Pupunzi" Bicocchi。插件的网址是：pupunzi.com.#mb.components/mb.miniAudioPlayer/miniAudioPlayer.html。GitHub 下载网址是：github.com/pupunzi/jquery.mb.miniAudioPlayer。
- ❑ 如对音频插件有疑问，请访问作者的支持网站 http://jquery.pupunzi.com。

第 17 章

为移动设备编写脚本

网络开发领域最新的重大机遇（同时也是挑战）皆源自移动设备越来越流行。现如今手机和平板电脑成了网上冲浪（以及更为重要的在线购买）的重要阵地，是时候积极面对移动设备了。本章将介绍如何处理设备的屏幕旋转与定位，从浏览器启动其他应用程序，以及针对特定设备编写代码。

17.1 改变方向

与针对台式机编程相比，为移动设备编程最大的区别在于设备的屏幕方向可以改变，也就是说，用户可以将手机从横屏持握转为竖屏持握，或者从竖屏持握转为横屏持握。在创建供访客通过台式机浏览的网站时，开发者不需要考虑方向的问题；而在创建移动网站时，这个问题比较重要。

例如，这个网站在最初设计的时候没有将移动设备考虑在内。现在检查它在移动设备上的呈现效果，看起来如图 17-1（竖屏）和图 17-2（横屏）所示。前一种情况还过得去，但是为了在一屏里多显示些内容，字号可以再调小一些（见图 17-3）。后一种情况字号就太大了，用户甚至需要手动缩小（见图 17-4）。下面是具体的做法。

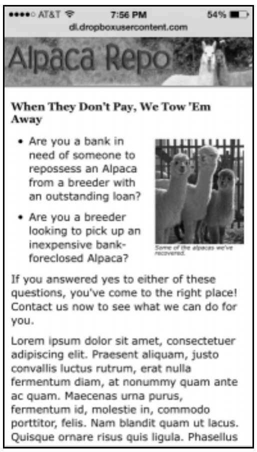


图 17-1　最初的页面，竖屏

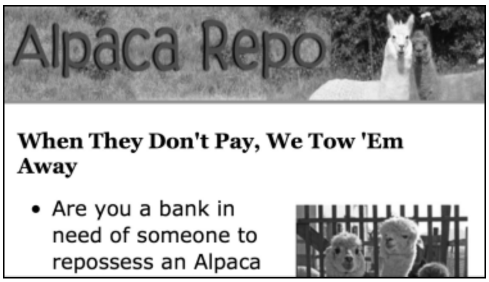


图 17-2　最初的页面，现在改为了横屏



图 17-3　调整字体大小后的竖屏页面



图 17-4　调整字体大小后，横屏页面显示的内容明显变多了

脚本 17-1　一个普通的公司网页

```
<!DOCTYPE html>
<html>
<head>
     <title>Welcome to Alpaca Repo</title>
     <meta name="viewport" content="width=560">
     <link rel="stylesheet" href="script01.css">
     <script src="script01.js"></script>
</head>
<body>
     <main>
        <header>
           <img src="images/header.jpg" alt="header">
        </header>
        <article>
           <h2>When They Don't Pay, We Tow 'Em Away</h2>
           <p class="photo"><img src="images/3alpacas.jpg" alt="alpacas">Some of the alpacas we've
           →recovered.</p>
           <ul>
              <li><p>Are you a bank in need of someone to repossess an Alpaca from a breeder with
              →an outstanding loan?</p></li>
              <li><p>Are you a breeder looking to pick up an inexpensive bank-foreclosed Alpaca?
              →</p></li>
           </ul>
           <p>If you answered yes to either of these questions, you've come to the right place!
           →Contact us now to see what we can do for you.</p>
           <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Praesent aliquam,  justo
           →convallis luctus rutrum, erat nulla fermentum diam, at nonummy quam  ante ac quam.
           →Maecenas urna purus, fermentum id, molestie in, commodo  porttitor, felis. Nam blandit
           →quam ut lacus. Quisque ornare risus quis  ligula. Phasellus tristique purus a augue
           →condimentum adipiscing. Aenean  sagittis. Etiam leo pede, rhoncus venenatis, tristique
           →in, vulputate at,  odio. Donec et ipsum et sapien vehicula nonummy. Suspendisse
           →potenti. Fusce  varius urna id quam. Sed neque mi, varius eget, tincidunt nec, suscipit
           →id, libero. In eget purus. Vestibulum ut nisl. Donec eu mi sed turpis feugiat  feugiat.
           →Integer turpis arcu, pellentesque eget, cursus et, fermentum ut, sapien. Fusce metus
           →mi, eleifend sollicitudin, molestie id, varius et, nibh.  Donec nec libero.</p>
```

```
            <h2>H2 level heading</h2>
            <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Praesent aliquam,  justo
            →convallis luctus rutrum, erat nulla fermentum diam, at nonummy quam  ante ac quam.
            →Maecenas urna purus, fermentum id, molestie in, commodo  porttitor, felis. Nam blandit
            →quam ut lacus.Quisque ornare risus quis  ligula. Phasellus tristique purus a augue
            →condimentum adipiscing.Aenean  sagittis. Etiam leo pede, rhoncus venenatis, tristique
            →in, vulputate at, odio.</p>
         </article>
         <footer>
            <p>Copyright &copy; 2009-2014 Alpaca Repo. All rights reserved.</p>
         </footer>
      </main>
</body>
</html>
```

脚本 17-2　添加一些自定义的 CSS

```
body {
     font-family: Verdana, Helvetica, Arial, sans-serif;
     background-color: #666;
     color: #000;
     margin: 0;
     padding: 0;
     text-align: center;
}

h1, h2, h3, h4, h5, h6 {
     font-family: Cambria, "Palatino Linotype", "Book Antiqua", "URW Palladio L", serif;
}

main {
     width: 80%;
     margin: 0 auto;
     padding: 0;
     text-align: left;
}

header {
     background-color: #91A43D;
     text-align: center;
     margin: 0;
     padding: 0;
     overflow: hidden;
}

header img {
     width: 100%;
     max-width: 800px;
     height: 110px;
}

article {
     background-color: #FFF;
     padding: 10px 20px 0 1em;
}

article::after {
     clear: both;
     content: "";
     display: table;
}
```

```
.photo {
     float: right;
     margin-left: .8em;
     width: 200px;
     font-size: .8em;
     font-style: italic;
     line-height: 1em;
}

footer p {
     margin: 0;
     padding: 10px 0;
     background-color: #DDD;
     text-align: center;
     font-size: .8em;
}

@media only screen and (max-device-width: 1024px) {
     main {
        width: 100%;
     }

     nav {
        font-weight: normal;
     }

     h1, h2, h3, h4, h5, h6 {
        font-family: Albany, Georgia, "Times New Roman", serif;
     }

     .landscape {
        font-size: .5em;
     }

     .portrait {
        font-size: .8em;
     }
}

@media only screen and (min-resolution: 192dpi), only screen and (webkit-min-device-pixel-ratio:2),
→only screen and (min-device-width:768px) and (min-device-height:768px) and (max-device-width:1024px) {
     header img {
        width: 640px;
     }
}
```

脚本 17-3　再添加一些 JavaScript 代码，响应式设计让页面变得更具有吸引力

```
addEventListener("load",resetPage,false);
addEventListener("orientationchange", resetPage,false);

function resetPage() {
     if (Math.abs(window.orientation) == 90) {
        classVal = "landscape";
     }
     else {
        classVal = "portrait";
     }
     document.getElementsByTagName("main")[0].setAttribute("class", classVal);
}
```

⇛ 处理方向改变

1.
```
@media only screen and (max-device-width:1024px) {
    main {
      width: 100%;
    }
    nav {
      font-weight: normal;
    }
    h1, h2, h3, h4, h5, h6 {
      font-family: Albany, Georgia,"Times New Roman", serif;
    }
    .landscape {
      font-size: .5em;
    }
    .portrait {
      font-size: .8em;
    }
  }
```

这里的 HTML（见脚本 17-1）和 CSS（见脚本 17-2）都很常见，除了上面的这段 CSS。我们关心的是最后两条规则，根据应用`.landscape`或者`.portrait`类来确定`font-size`（字体大小）。不过，你会发现这两个类都没有用在 HTML 中，而是由 JavaScript 设置。

2.
```
addEventListener("load",resetPage,false);
addEventListener("orientationchange",resetPage,false);
```

在 JavaScript 代码（脚本 17-3）中，我们首先添加了两个事件处理程序：`load`（前面已经出现过多次）和`orientationchange`。两者被触发都会调用`resetPage()`函数。

3.
```
if (Math.abs(window.orientation) == 90) {
    classVal = "landscape";
  }
  else {
    classVal = "portrait";
  }
```

`window`对象的`orientation`属性有 4 个值：0（竖屏，顶部朝上）、90（横屏，向左）、180（竖屏，顶部朝下）和–90（横屏，向右）。

在`restePage()`函数中，如果`window.orientation`的绝对值为 90，那么我们知道设备处于横屏模式，不管是向左还是向右；否则处于竖屏模式。对于这两种情况，我们都相应地设置`classVal`的值。

4.
```
document.getElementsByTagName("main")[0].setAttribute("class",classVal);
```

最后，我们将存储的`classVal`值作为`main`标签的一个`class`属性。

✔提示

- ❑ 第 1 步中`@media`查询的作用是只有运行在移动设备上时才应用这里的 CSS。脚本 17-2 最后的`@media`查询限定，只有运行在高分辨率移动设备上时才应用此处的 CSS。
- ❑ `@media`查询不在本书介绍范围之内，如果想进一步了解`@media`查询，可以查阅 HTML 或 CSS 参考书（我们的选择是 Elizabeth Castro 和 Bruce Hyslop 合著的《HTML5 与 CSS3 基础教程（第 8 版）》）。

调试移动设备

当开发人员调试网站的时候，他们手边不太可能备齐所有的 iOS 和 Android 设备。所幸苹果和谷歌分别提供了 Android 模拟器和 iOS 模拟器，为大家省下了一大笔开销，要知道各种型号的手机和平板电脑烧钱不少呢。下载两家公司提供的 SDK（软件开发工具包）会自带模拟器。

- **Android 模拟器**：可从 developer.android.com/sdk 上下载跨平台的 Android SDK。要详细了解该模拟器的工作原理，可访问 developer.android.com/tools/devices/emulator.html。
- **iOS 模拟器**：苹果公司将其模拟器与 Xcode 开发者工具绑在了一起。可从 developer.apple.com/xcode/downloads/下载到 Xcode。iOS 模拟器的用法很简单，但很可惜，它只能在 Mac 上使用。

这两个模拟器都是免费的。

✓提示

- 在 iOS 上调试网页有一条非常棒的技巧：在 iOS 模拟器中查看网页时运行 Safari。切换到 Safari，查看 Develop（开发）菜单，会看到一个新选项：iPhone（或 iOS）Simulator（见图 17-5）。这样，你就可以在加载到模拟器中的页面上使用 Safari 的开发工具了（如 Web Inspector）。

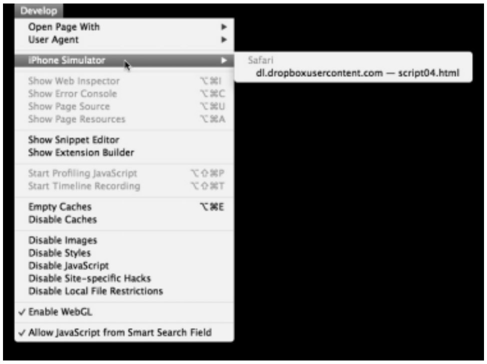


图 17-5　有了这个方便的小菜单项，就能够使用台式机上的全套工具调试移动页面

表17-1　移动事件

事　　件	触发动作
gesturestart	两个或多个手指触摸屏幕
gesturechange	手指触屏位置发生变化
gestureend	结束手势
orientationChange	旋转设备
touchstart	手指触摸屏幕
touchmove	手指在触屏时移动
touchend	结束触摸
touchcancel	浏览器自身取消触摸追踪

表17-2 触摸属性

属性	含义
clientX clientY	触摸位置相对于窗口视口的坐标
identifier	当前触摸的唯一标识符
pageX pageY	触摸位置在页面坐标系中的坐标
screenX screenY	触摸位置在屏幕坐标系中的坐标
target	触摸的目标

表17-3 GestureEvent对象

属性	方法
altKey	initGestureEvent
ctrlKey	
metaKey	
rotation	
scale	
shiftKey	
target	

表17-4 TouchEvent对象

属性	方法
altKey	initTouchEvent
ctrlKey	
metaKey	
rotation	
scale	
shiftKey	
targetTouches	
changedTouches	
touches	

表17-5 DeviceMotionEvent对象

属性	方法
acceleration	initDeviceMotionEvent
accelerationIncludingGravity	
interval	
rotationRate	

表17-6 DeviceOrientationEvent对象

属性	方法
alpha	initDeviceOrientationEvent
beta	
gamma	
webkitCompassAccuracy	
webkitCompassHeading	

17.2　处理触摸事件

你很快就会发现，为移动设备进行开发常常需要采取一些折中策略。一个常见的例子是，放弃使用悬停事件，而采用触摸事件。这个小示例允许我们通过触屏随意移动页面中的元素。脚本 17-4 为这个例子的 HTML 代码，脚本 17-5 为 CSS 代码，脚本 17-6 为 JavaScript 代码。

脚本 17-4　这个 HTML 页面非常单调无趣

```
<!DOCTYPE html>
<html>
<head>
     <title>Drag n' drop touch demo</title>
     <meta name="viewport" content="width=device-width">
     <link rel="stylesheet" href="script02.css">
     <script src="script02.js"></script>
</head>
<body>
     <div id="draggable">
        Drag me!
        <p id="boxLocation"></p>
     </div>
</body>
</html>
```

脚本 17-5　少量样式设置让它看起来更吸引人

```
#draggable {
     position: absolute;
     background-color: #FFC;
     border: 5px solid yellow;
     width: 100px;
     height: 100px;
     padding: 50px;
     text-align: center;
     font-family: Albany, Georgia, "Times New Roman", serif;
}
```

脚本 17-6　移动元素，效果形象生动

```
window.addEventListener(
     "load",
     function() {
        document.getElementById("draggable").addEventListener("touchmove",moved,false);
     },
     false
);

function moved(evt) {
     evt.preventDefault();
     var dragBox = document.getElementById("draggable");

     dragBox.style.left = (evt.changedTouches[0].pageX - 100) + "px";
     dragBox.style.top = (evt.changedTouches[0].pageY - 100) + "px";

     document.getElementById("boxLocation").innerHTML = "Top: " + dragBox.style.top + "<br>
     ➝Left: " + dragBox.style.left;
}
```

⇛ 处理触摸事件

1. window.addEventListener(
```
    "load",
    function() {
        document.getElementById("draggable").addEventListener("touchmove",moved,false);
    },
    false
);
```

初次加载页面（见图 17-6）我们希望给 id 为 draggable 的元素添加一个事件处理程序。新事件 touchmove 会在每次被触发（用户滑动元素）时调用 moved()函数。



图 17-6　初次加载页面（在 Android 设备中显示）

2. evt.preventDefault();
```
var dragBox = document.getElementById("draggable");
```

通常，用户在屏幕上滑动手指会触发一个选择。我们不希望这样，因此 moved()函数在开始时就阻止了默认行为。接下来创建一个新变量 dragBox，由此我们不必不停地输入 document.getElementById。

3. dragBox.style.left = (evt.changedTouches[0].pageX - 100) + "px";
```
dragBox.style.top = (evt.changedTouches[0].pageY - 100) + "px";
```

重置盒子的 top 和 left 属性实际上就是移动盒子。乍一看有些不好理解，容我们慢慢道来。

重置 top 和 left 可以移动元素的左上角，但很有可能这并不是手指（或触控笔）真正触屏的位置。如果将元素的左上角重置为手指触屏的位置，看起来元素的移动就会显得过于跳跃。 17

手指更有可能指向元素中心。要基于元素中心的运动来移动左上角，我们需要从左端位置与顶端位置（即横坐标与纵坐标）中各减去 100 像素。这两个位置分别对应于 evt.changedTouches[0].pageX

和 evt.changedTouches[0].pageY。

4. document.getElementById("boxLocation").innerHTML = "Top: " + dragBox.style.top +
→"
Left: " + dragBox.style.left;

最后，为了让大家看清楚我们确实移动了元素，图 17-7 中显示了元素左上角的当前位置。



图 17-7　这是将元素摆弄一阵后的样子

✔提示

- 如果对重置元素位置的方法不太熟悉，可以看看脚本 8-9 和脚本 11-14。

17.3　针对不同设备编写特定代码

不同设备的制造商不同，运行不同的系统，使用不同的浏览器，因此它们有时会表现不同。这样的结果是，我们有时候需要针对某些设备编写代码。这个例子中，我们针对的是 Android 设备。

脚本 17-7　添加少量代码，只针对 Android 设备添加样式

```
window.addEventListener(
     "load",
     function() {
        document.getElementById("draggable").addEventListener("touchmove",moved,false);
        androidSS();
     },
     false
);

function androidSS() {
     if (navigator.userAgent.match(/android/i)) {
        var fileref = document.createElement("link");
        fileref.setAttribute("rel","stylesheet");
```

```
            fileref.setAttribute("href","script03.css");
            document.getElementsByTagName("head")[0].appendChild(fileref);
        }
    }

    function moved(evt) {
        evt.preventDefault();
        var dragBox = document.getElementById("draggable");

        dragBox.style.left = (evt.changedTouches[0].pageX - 100) + "px";
        dragBox.style.top = (evt.changedTouches[0].pageY - 100) + "px";

        document.getElementById("boxLocation").innerHTML = "Top: " + dragBox.style.top + "<br>Left: " +
        ➝dragBox.style.left;
    }
```

脚本 17-8 这个样式表只会在 Android 设备中加载

```
#draggable {
    font-family: "Droid Sans", sans-serif;
}
```

⇛ 针对不同设备编写特定代码

1.
```
window.addEventListener(
    "load",
    function() {
      document.getElementById("draggable").addEventListener("touchmove",moved,false);
      androidSS();
    },
    False
  );
```

这个任务的 HTML 与 CSS 与上一个例子完全相同，但现在我们还希望，仅在用户使用 Android 设备时加载另一个样式表。这由 `androidSS()`函数来处理，它会在页面最初加载时被调用，如脚本 17-7 所示。

2.
```
function androidSS() {
    if (navigator.userAgent.match(/android/i)) {
      var fileref = document.createElement("link");
      fileref.setAttribute("rel","stylesheet");
      fileref.setAttribute("href","script03.css");
      document.getElementsByTagName("head")[0].appendChild(fileref);
    }
  }
```

`navigator` 对象有一个名为 `userAgent` 的属性，该属性包含一个描述设备的字符串。如果 `userAgent` 属性含有字符串`"android"`，我们会添加指向另一个样式表的链接，如脚本 17-8 所示。Android 用户现在会看到可拖动框中无衬线字体的文本（见图 17-8）。



图 17-8　同样的页面，新的样式

✔提示

- 这个技术不限于引入额外的样式表，它还可以用于有针对性地编写脚本。
- 目前而言，很遗憾，还没有适用于所有移动设备的字体。要了解有哪些字体可用，建议查看 jordanm.co.uk/tinytype 上的兼容列表，该列表涵盖了 iOS、Android、Windows Phone 和 BlackBerry 设备。例如，Android 设备出厂时只有 4 种字体：Droid Sans、Droid Serif、Droid Sans Mono 和 Roboto。

17.4　定位设备

移动设备的一大便捷之处是我们可以用它来确定自己的当前位置，甚至可以为网站定位（需要少量 JavaScript 代码）。重要的是不仅能确定我们在哪里，还可以启动其他应用，即在浏览器外部而不是内部启动地图。

脚本 17-9　这个简单的 HTML 页面含有一个作用于移动设备的强大功能：启动另一个应用

```
<!DOCTYPE html>
<html>
<head>
     <title>Form example</title>
     <meta name="viewport" content="width=device-width">
     <script src="script04.js"></script>
 </head>
 <body>
      <p>
        <input type="text" pattern="[0-9]*" placeholder="Zip code"><br>
        <input type="email" placeholder= "Enter your email address" size="40">
     </p>
     <p>
```

```
        Lat: <input type="text" id="lat_field" name="latitude"><br>
        Lng: <input type="text" id="lng_field" name="longitude">
    </p>
    <a href="http://maps.apple.com?z=16&amp;ll=" id="mapQuery">View address on map</a>
</body>
</html>
```

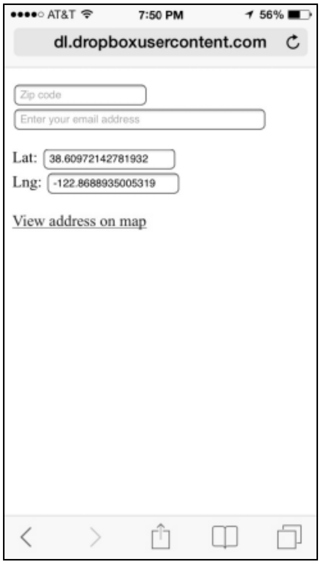


图 17-9　这里是我们当前的纬度和经度，但真正重要的是点击链接后的结果



图 17-10　iPhone 上苹果公司的地图应用很好地显示了作者的当前位置

脚本 17-10　这段 JavaScript 代码填写纬度和经度字段，然后建立地图链接

```
window.addEventListener("load",getLocation,false);

function getLocation() {
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(
            function(position) {
                document.getElementById("lat_field").value = position.coords.latitude;
                document.getElementById("lng_field").value = position.coords.longitude;

                document.getElementById("mapQuery").href += position.coords.latitude + "," +
                ➝position.coords.longitude;
            }
        );
    }
}
```

⇛ 进行定位

1. View address on map

脚本 17-9 包含了建立图 17-9 中的链接所需的 HTML 代码。点击该链接会自动启动设备的地图应用，如图 17-10 所示（这里是苹果公司的地图应用）。

2. `if (navigator.geolocation) {`
```
    navigator.geolocation.getCurrentPosition(
```
在脚本 17-10 中，`getLocation()`函数在页面加载时被调用。该函数首先检查 `navigator` 对象是否含有 `geolocation` 属性，如果含有，则调用该属性的 `getCurrentPosition()`函数。

3. `function(position) {`
```
    document.getElementById("lat_field").value = position.coords.latitude;
    document.getElementById("lng_field").value = position.coords.longitude;
```
将带 `position` 参数的匿名函数传给 `getCurrentPosition()`函数。使用 `position` 参数，这段脚本可以获得我们的经度和纬度，并将其显示在页面上（见图 17-9）。

4. `document.getElementById("mapQuery").href += position.coords.latitude + "," + position.` `→coords.longitude;`

这段脚本最后将经度和纬度值附加到地图查询链接上，从而精准地锁定了我们的当前位置。

✔提示

- 在第 1 步中，`ll` 显然代表纬度和经度。`z` 代表缩放（zoom），即你希望将地图缩小或扩大到什么程度。`z` 值越大，地图越详细。
- 好消息是，苹果和谷歌的地图服务都使用同样的 `z` 和 `ll` 属性。
- 坏消息是，没有一个可以用于所有设备的地图 URL。第 1 步中的 URL（maps.apple.com）只适用于 iOS 设备（如 iPhone 和 iPad）。在 Android 设备上，你需要使用前一个任务中的流程加载 maps.google.com。这并不是说 maps.google.com 不能用在 iOS 设备上，但要想使用，首先需要安装谷歌地图应用。这一点并非你所能掌控的，因此要在代码中处理这两种情形。
- 页面初次加载会弹出一个提示对话框，询问是否允许网站获取设备的当前位置数据。用户必须允许此功能才能进行定位。
- 表 17-7 给出了其他的一些应用示例，这些应用可以用不同的链接目标来启动。

表17-7　启动移动应用

href值	结　果
`mailto:me@example.com`	启动Mail应用
`facetime:me@example.com`	启动FaceTime应用并开始聊天（仅限于iOS）
`tel:555-1212`	切换到电话应用并拨号
`sms:555-1212`	启动短信应用（如iOS上的短信应用）
`http://www.youtube.com`	启动YouTube应用（如果可用）

第 18 章 bookmarklet

你知道，可以使用 JavaScript 从网页内部控制 Web 浏览器。但是，在不使用网页的情况下，也可以使用 JavaScript 控制浏览器，所用的技术称为 bookmarklet。bookmarklet 是一种特殊的书签（用 IE 的术语来说是收藏夹，有时候 bookmarklet 也称为 favelet），其中包含对浏览器的 JavaScript 解释器的调用，而不是外部 URL。bookmarklet 中的 JavaScript 可以做任何事情，比如获得图像的相关信息，给出一个单词的定义以及调整浏览器窗口。如果你掌握了 JavaScript，就很容易添加这种功能，从而使你的浏览器成为更智能化、更好的工具。

bookmarklet 与其他 JavaScript 代码不太一样，因为它们有一个重要而有意思的格式限制：它们必须写在一行中。要使用分号把命令连在一起。

在本章中，我们将介绍一些有用的 bookmarklet，只需稍费点儿工夫，你就能够编写出自己的 bookmarklet。

18.1 第一个 bookmarklet

脚本 18-1 确实不是个很让人兴奋的脚本，它是我们常见的 Hello World 脚本的变体。但重要的是，你让 Web 浏览器执行了某些操作，而没有加载网页。这个示例还演示了如何在各种浏览器中创建和使用 bookmarklet。

脚本 18-1 这个 bookmarklet 返回 Hello World

```
javascript:alert('Hello World');
```

⇛ 在 Firefox 中创建 bookmarklet

1. 在 Bookmarks 菜单上，选择 Show All Bookmarks（见图 18-1）。Library 窗口打开。

2. 在左栏中，选择 Bookmarks Toolbar。在上面的工具栏中，选择 Action 菜单（在 Mac 中，见图 18-2）或 Organize 菜单（在 Windows 中，见图 18-3），然后选择 New Bookmark。

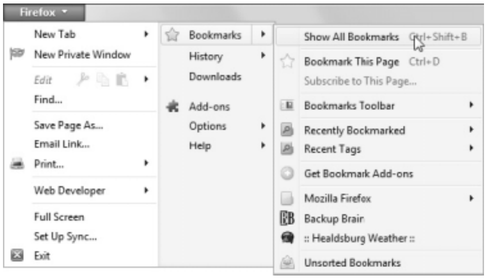


图 18-1　Firefox（图中是 Windows 版本，但是 Mac 版本是相似的）允许用户选择 Show All Bookmarks，在 Library 中输入 bookmarklets



图 18-2　苹果机上的 Firefox 在 Action 菜单中提供 New Bookmark 菜单项

3. 在 Name 字段中，输入希望出现在工具栏上的名称。在这个示例中，输入 Hello。

4. 在 Location 字段中（见图 18-4），输入 `javascript:alert('Hello World');`，然后单击 Add。

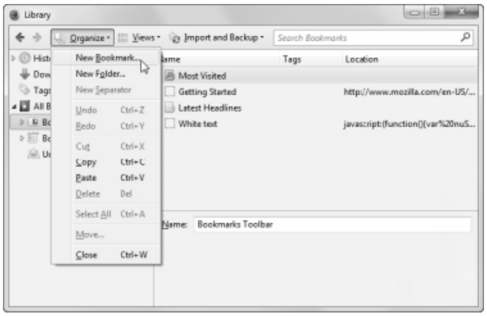


图 18-3　Windows 上的 Firefox 在 Organize 菜单中提供 New Bookmark 菜单项

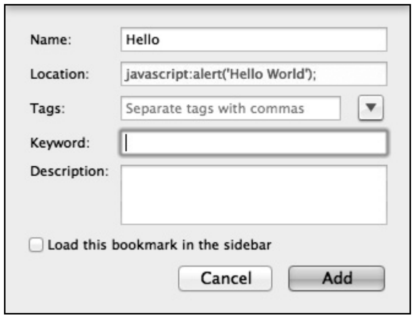


图 18-4　在 Firefox 中，在 New Bookmark 对话框中向 Location 字段添加 bookmarklet 代码

这时会返回到 Library 窗口。

5. 关闭 Library 窗口。新按钮应该出现在 Bookmarks Toolbar 上。单击这个按钮就会激活命令，显示一个包含文本“Hello World”的警告框。

✔提示

❑ 如果在 Firefox 中没有看到 Bookmarks Toolbar，那么可以通过选择 View→Toolbars→Bookmarks Toolbar 来显示它。

⇛ 在 Safari 中创建 bookmarklet

1. 确保 View 菜单中的菜单项显示 Hide Favorite Bar 而不是 Show Favorite Bar，从而保证 Favorites Bar 可见。

2. 在 Address Bar 中，输入 `javascript:alert('Hello World');`。

3. 将地球图标（输入文本的左边）从 Address Bar 拖动到 Favorites Bar 上，并且释放鼠标按钮。此时出现一个文本字段，要求我们输入新书签的名称（见图 18-5）。



图 18-5　在 Address Bar 中输入 bookmarklet 代码，并且将它拖动到 Favorites Bar 上，Safari 就会提示输入 bookmarklet 的名称

4. 输入 bookmarklet 的名称。

新的 bookmarklet 以按钮形式出现在 Favorites Bar 中，单击这个按钮就会激活命令。

✔提示

- ❑ 基于自己的偏好，也可以选择 Bookmarks→Edit Bookmarks，将 bookmarklet 从 Favorites Bar 转移到 Bookmarks 菜单中。
- ❑ 要想从 Safari Favorites Bar 中删除 bookmarklet，应该将它拖动到浏览器窗口中。它会显示一段烟雾动画，然后就消失了。如果意外地删除了不该删除的书签，可以按 Cmd-Z 来恢复它。

bookmarklet 的起源

bookmarklet 的想法最初来自 Netscape JavaScript Guide，它指出如何在 Personal Toolbar 中添加 JavaScript。现在主持 Bookmarklets.com 站点的 Steve Kangas 发明了术语 bookmarklet。他的 Web 站点（www.bookmarklets.com）虽然近几年没怎么变化，但包含数百个有用的 bookmarklet，本章只涉及一些皮毛。得到他的允许，本章中的一些示例参考了该站点上的脚本。

⇛ 在 Chrome 中创建 bookmarklet

1. 右击 Bookmarks 菜单栏，选择 Add Page（见图 18-6），出现 Add Page 对话框。
2. 确保下面的 Bookmarks Bar 高亮显示，然后在 Name 字段输入 Hello。
3. 在 URL 字段输入 `javascript:alert('Hello World');`（如图 18-7 所示），并单击 Save。此时新的 bookmarklet 就以按钮的形式出现在 Bookmarks Bar 中。单击这个按钮就会激活命令。

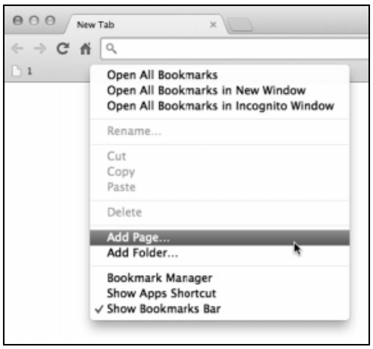


图 18-6　在 Chrome 中，从 Bookmarks 菜单栏中选择“添加网页”即可添加 bookmarklet

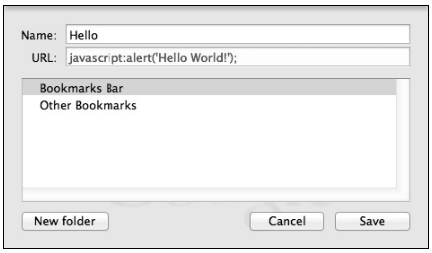


图 18-7　在 Chrome 对话框的 URL 字段输入 bookmarklet 的代码

✔提示

- ❑ Chrome 有个特别怪异的行为：必须在网页上才能运行 bookmarklet。也就是说，普通的 New Tab 页面无法打开 bookmarklet，用户必须首先打个某个网页，任何网页都行。

⇛ 在 IE 中创建 bookmarklet

1. 在 Favorites 菜单中，选择 Add to Favorites。Add a Favorite 对话框出现（见图 18-8）。

2. 在 Name 字段中，输入 Hello。

3. 在对话框的 Create in 部分中，选择 Favorites Bar 文件夹以确保新的书签出现在 Favorites Toolbar 中，单击 Add 按钮。

4. 右击 Links Toolbar 中新的“Hello”bookmarklet，选择 Properties。Properties 对话框出现。在 URL 字段中，输入 `javascript:alert('Hello World');`（见图 18-9），如果我们对 IE 分配给按钮的图标满意（它往往是当前网页的图标），那么单击 OK 按钮。



图 18-8　这里是 Add a Favorite 对话框

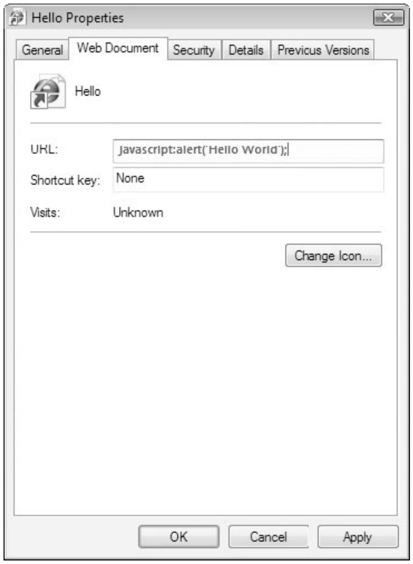


图 18-9　再次修改属性，将 JavaScript 代码添加到 bookmarklet

5.（可选）如果希望改变图标，那么单击 Properties 对话框中的 Change Icon。Change Icon 对话框出现，选择我们需要的图标，单击 OK 按钮，然后再次单击 OK 按钮以关闭 Properties 对话框。

6. 如果适当地设置了 IE 的安全设置，那么应该会出现 Problem with Shortcut 对话框。单击 Yes 按钮。

7. 单击 Links Toolbar 上的 Hello 按钮来激活命令。

✔提示

- ❑ 默认情况下，IE 会隐藏 Favorites Bar。要显示它，需右击窗口顶部的工具栏，并在弹出的快捷菜单中选择 Favorites Bar。

bookmarklet 与 IE 安全性

微软 Windows 的 IE 的每个版本都有许多安全问题，这并不是什么秘密。微软以各种各样的方式解决这些问题。Windows XP Service Pack 2 主要致力于提高整个 Windows，包括 IE 6 的安全性（顺便说一句，如果你仍在使用 IE 6，建议你立即升级）。IE 7 会集成更多的安全特性，IE 7 最初计划在 Windows Vista 之后发布，但是 Vista 的发布日期多次推迟，IE 7 提前发布了。Vista 增加了一些安全特性，Windows 和 IE 的后续版本进一步增强了安全性。

我们的测试表明，bookmarklet 的表现在各版本之间是不一致的。我们发现，一些 bookmarklet 只能在某些页面上运行，还有一些 bookmarklet 根本无法运行，具体情况取决于在 IE 中启用了哪些安全设置。这些设置包括是否打开了弹出窗口拦截，以及 Tools→Internet Options 的 Security 选项卡中的许多设置。简单地说，由于在 IE 中添加了更多的安全层，bookmarklet 在 IE 中运行起来更困难了。通过修改安全设置，可能会使更多的 bookmarklet 能够在 IE 中运行，但是因为 IE 在安全问题方面的记录很糟糕，我们不建议你草率地修改安全设置。

我们的建议是：如果必须在降低 IE 的安全程度和不能运行 bookmarklet 之间作出选择，那么不如改用 Firefox、Safari 或者 Chrome。这会解决许多浏览器安全问题，而且会得到一致的 bookmarklet 功能。

18.2 改变页面的背景颜色

脚本 18-2 很简单，但是功能强大。你是否遇到过这样的情况：你访问的一个站点包含许多有用的信息，但是站点的背景颜色与文本颜色太接近，以致信息很难阅读？更糟糕的情况是，页面的作者使用很刺眼的颜色，使你的眼睛很疲劳。这个小小的 bookmarklet 就能够解决这些问题。注意，在这种情况下，是使用 bookmarklet 改变你查看别人页面的方式——这正是 bookmarklet 的特殊之处。当然，它不会改变实际的页面，仅仅是改变浏览器显示页面的方式。

脚本 18-2 这个脚本将背景颜色改为白色，它会大大改进设计不当的页面的显示效果

```
javascript:void(document.body.style.background='#FFF');
```

⇛ 改变页面的背景颜色

- ❑ `javascript:void(document.body.style.background='#FFF');`

这个脚本使用 `document.body.style.background` 对象并且将它重新设置为白色。现在，我们可以清楚地看到页面上的文本了，如图 18-10 和图 18-11 所示。

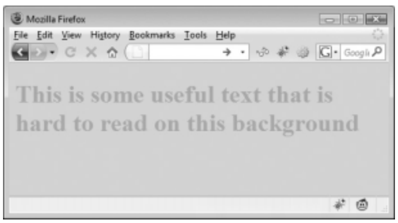


图 18-10 在页面原来的背景颜色上，文本很难看清楚

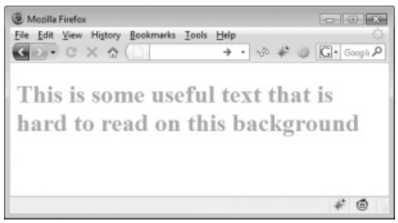


图 18-11 在白色背景上，文本清楚多了

✔提示

- ❑ 注意，这个 bookmarklet 使用了 `javascript:void(command);`形式。这是因为 bookmarklet 必须返回某个值，这个值一般用来覆盖当前页面的内容。通过使用 `void()`方法，就什么也不返回，什么也不覆盖。
- ❑ bookmarklet使用单引号,而不是双引号。这是因为在幕后每个bookmarklet包含在一个`<a href="">`标签中。使用双引号会过早地结束 bookmarklet。

18.3　改变页面样式

如果页面的背景颜色是在页面的 HTML 中设置的，那么前面的示例会起作用。但是，如果页面使用样式表改变背景颜色或者在页面元素上使用背景图像，那么这个 bookmarklet 就无效了。下一个 bookmarklet（见脚本 18-3）会更换页面背景颜色、文本颜色和链接颜色的 CSS 样式。背景颜色改为白色，文本颜色改为黑色，链接是蓝色的，已经访问过的链接是紫色的。在图 18-12 和图 18-13 中可以看到这些变化。

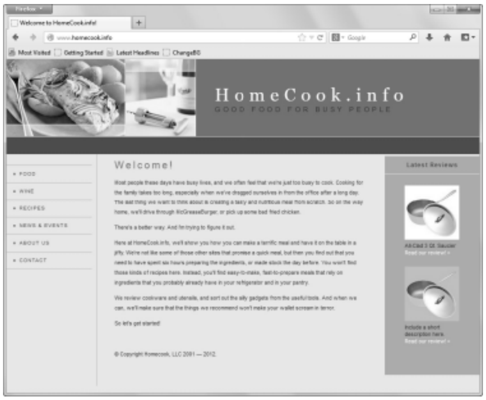


图 18-12　原来的页面比较难以阅读，所以我们希望改变 CSS 样式，使页面的可读性更好

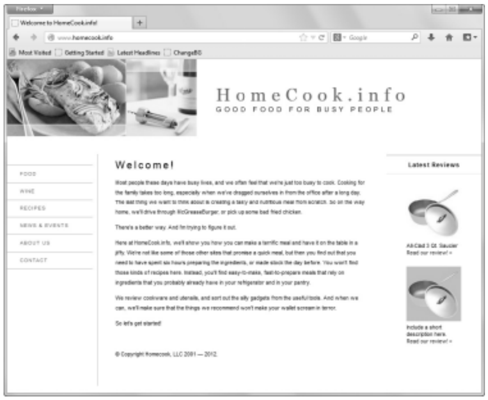


图 18-13　通过修改背景颜色和链接颜色，页面比较容易阅读了

脚本 18-3　不要误解箭头的意思，它们并不表示代码中有换行，而是因为本书的页面不够宽。请记住，实际的 bookmarklet 必须在一行上。这个 bookmarklet 会改变页面的样式，使页面的可读性更好

```
javascript:(function(){var styles='*{background:#FFF !important;color:#000 !important}:link,:link
→*{color:#00F !important}:visited,:visited *{color:#93C !important}';var newSS=document.create
→Element('link');newSS.rel='stylesheet';newSS.href='data:text/css,'+escape(styles);document.getElem
→entsByTagName('head')[0].appendChild(newSS);})();
```

如果你觉得阅读网页中黑色背景上的白色文本有困难，或者不喜欢站点对链接和一般文本使用太相似的颜色，那么这个 bookmarklet 正好可以帮助你。我们经常在自己的日常 Web 浏览中使用这个 bookmarklet。再次提醒你，这个 bookmarklet 只是改变页面在你浏览器中的外观，而不会改变页面本身。实际上，如果重新加载页面，它会重新显示为原来的难以阅读的样式。但是，通过使用这个

bookmarklet，只需单击一次按钮就可以减轻视觉负担。

⇛ 改变页面样式

1. `javascript:(function(){`

这里采用一种新的形式：并不使用 `void()`向浏览器返回 `null` 值，而是把整个 bookmarklet 放在一个匿名函数中。

2. `var styles = '* {background:#FFF !important;color:#000 !important}:link,:link *{color:`
`➝#00F !important}:visited,:visited *{color:#93C !important}';`

这一行在页面上创建一个新的变量 `styles`，它设置我们希望为页面添加的样式：背景白色，文本黑色，未访问的链接蓝色，已访问的链接紫色。`!important` 使这些规则覆盖所有其他样式，*表示这些新样式应用于页面上的所有元素。

3. `var newSS = document.createElement('link');`
`newSS.rel = 'stylesheet';`

这一行在页面上创建一个新的 `link` 元素，然后将这个元素存储在新变量 `newSS` 中。第二行为新创建的链接创建一个新的 `rel` 属性，然后将它的值设置为'stylesheet'。在这里，`rel` 属性告诉浏览器我们要链接一个样式表。

4. `newSS.href = 'data:text/css,' + escape(styles);`

这一行为新创建的 `link` 元素添加一个新的 `href` 属性，并将其设置为我们之前创建的样式。

5. `document.getElementsByTagName('head')[0].appendChild(newSS);`

这一行将新的链接元素插入网页，让新样式生效，使页面的可读性更强。

6. `})();`

这里结束上面创建的匿名函数（}），然后结束函数包装器（)），后面的()只是为了表示这个函数已经结束并让浏览器马上运行它。

✔提示

- ❑ 看起来这里的代码太复杂了，仅仅为了避免 `void()`似乎不值得。但是，这是很有意义的：JavaScript 认为 bookmarklet 是在当前页面内运行的；如果采用原来的方式，就无法确保页面本身并未使用 bookmarklet 中的变量。如果把整个 bookmarklet 放在一个函数中，变量就处于函数的范围内（参见第 2 章），可以避免变量冲突。
- ❑ 正如在本章开头提到的，一个 bookmarklet 只能有一行代码。在语句之间放上分号，就能够将所有命令放在一行上。

18.4 查询单词

如果使用 Web 浏览器进行写作（尤其是写邮件），你会希望有在大部分数字处理程序中可用的字典和词典工具。通过脚本 18-4 和脚本 18-5，就能够在所有写作工作中具备这种功能。我们将使用 bookmarklet 查询在线字典或词典。因为脚本非常相似，所以我们在同一节中一起介绍它们。脚本 18-4 演示如何进行字典查询，脚本 18-5 演示如何进行词典查询。

脚本 18-4　这个 bookmarklet 执行字典查询

```
javascript:(function(){var inText=window.getSelection()+'';if(!inText){inText=prompt('Word:','');}
→if(inText){window.open('http://www.answers.com/topic/'+escape(inText)+'#Ameircan_Heritage_
→Dictionary_ds','dictWin','width=800,height=500,left=75,top=175,s_crollbars=yes');}})();
```

脚本 18-5　这个 bookmarklet 执行词典查询

```
javascript:(function(){var inText=window.getSelection()+'';if(!inText){inText=prompt('Word:','');}
→if(inText){window.open('http://www.answers.com/topic/'+escape(inText)+'#Roget\'sThesaurusds',
→'thesWin','width=800,height=500,left=75,top=175,scrollbars=yes');}})();
```

⇒ 查询单词

1. `var inText = window.getSelection() + '';`

这一行创建一个新变量 inText，并且将它设置为浏览器中选择的文本的值。

注意，在第一行中，最后是两个单引号，而不是一个双引号。这么做是因为浏览器可能会返回非字符串内容，从而可以将结果转换为字符串。

2.
```
if (!inText){
  inText = prompt('Word:','');
}
```

如果没有选择任何文本，则提示用户输入要查询的单词。

3. `if (inText) {`

用户有了两次机会，所以他们应该已经输入了要查询的单词。即使如此，我们还是在执行查询之前进行检查。

4.
```
window.open('http://www.answers.com/topic/' + escape(inText) + '#Ameircan_Heritage_
→Dictionary_ds','dictWin','width=800,height=500,left=75,top=175,scrollbars=yes');
```

或者

```
window.open('http://www.answers.com/topic/' + escape(inText) + '#Roget\'s_Thesaurus_ds','
→thesWin','width=800,height=500,left=75,top=175,scrollbars=yes');
```

根据是要进行字典查询还是词典查询，选择这两段代码之一，如图 18-14 和图 18-15 所示。它们都打开一个新窗口，其中显示我们请求的信息。可以通过改变 window.open()调用的 height 和 width 属性，改变窗口的尺寸来适应自己的屏幕。



图 18-14　触发字典 bookmarklet 就会返回显示查询结果的窗口



图 18-15　词典查询的结果

✔提示

- ❑ 尽管与 JavaScript 没什么关系，但是有一点很有意思，值得注意：如果使用运行 OS X 10.4 或更高版本的 Mac 机，那么在 Safari 或许多其他程序中，可以将鼠标指针放在一个单词上，并且按 Cmd-Ctrl-D，操作系统就会利用 Dictionary 应用程序弹出一个字典/词典窗口（见图 18-16）。这在任何基于 Cocoa 的程序中都有效，所以如果在 Mac 上使用 Firefox，就没有这种功能（倒霉）。如果你不知道“基于 Cocoa 的程序”是什么意思，则不必担心，在你所用的程序中试试这种功能就行了。

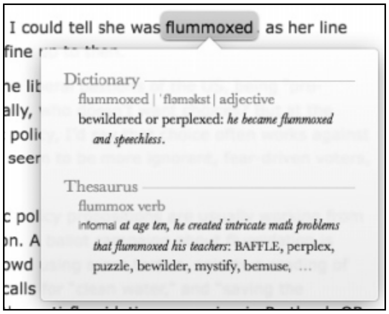


图 18-16　在 Mac OS X 中，内置了一个字典和词典

18.5　查看图像

对于设计人员有帮助的一种功能是，脱离页面的布局，查看页面上的所有图像。脚本 18-6 可以探察别人的页面，并且显示页面上的各个图像、图像（在现代浏览器中）的高度和宽度以及它们的 URL。

脚本 18-6　可以用这个脚本查看页面图像表格

```
javascript:(function(){var iWin,i,t='',di=document.images;for(i=0;i<di.length;i++){if(t.indexOf
→(di[i].src)<0){t+='<tr><td><img src='+di[i].src+'/></td><td>'+di[i].height+'</td><td>'+di[i].
→width+'</td><td>'+di[i].src+'</td></tr>';}}if(t==''){alert('No images!');}else{iWin=window.open
→('','IW','width=800,height=600,scrollbars=yes');iWin.document.body.innerHTML='<table border=1
→cellpadding=10 cellspacing=0><tr><th>Image</th><th>Height</th><th>Width</th><th>URL</th></tr>'+
→t+'</table>';}})();
```

⇒ 查看图像

```
1. var iWin,i,t = '',di=document.images;
```

这个 bookmarklet 首先对 4 个变量进行初始化：iWin、i、t（它以后将包含所有输出）和 di（它包含 document.images 对象）。

```
2. for (i=0;i<di.length;i++) {
```

现在循环遍历文档中的每个图像。

```
3. if (t.indexOf(di[i].src)<0) {
```

在这一步中，检查是否已经将图像放在页面上了。这行代码防止一个图像出现一次以上。

```
4. t +='<tr><td><img src=' + di[i].src + '/></td><td>' + di[i].height + '</td><td>' + di[i].width
  →+'</td><td>' + di[i].src + '</td></tr>';}}
```

我们需要的所有信息都在这里以漂亮的表格格式写出。第一个单元格包含图像，第二个单元格包含高度，第三个包含宽度，最后一个包含图像的 URL。

```
5. if (t=='') {
    alert('No images!');
  }
```

当循环完成时，检查是否已经找到了任何图像。如果没有找到任何图像，那么显示一个警告窗口，向用户显示“No images!”。

```
6. else{
    iWin = window.open('','IW','width=800,height=600,scrollbars=yes');
```

如果找到了图像，就打开一个新窗口来显示图像信息。

```
7. iWin.document.body.innerHTML = '<table border=1 cellpadding=10 cellspacing=0><tr><th>Image
  →</th><th>Height</th><th>Width</th><th>URL</th></tr>'+t+'</table>';
```

这里创建和显示新窗口。图像信息以及每列的标题信息被写出，效果见图 18-17。

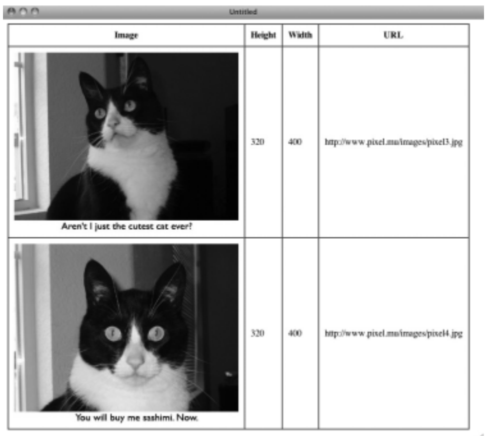


图 18-17　这个脚本将图像和信息显示在一个吸引人的表格中

18.6 显示 ISO Latin 字符

如果你手工编写网页，要记住像 á 和 à 这类特殊字符的编码不容易。脚本 18-7 可以显示常用的元音字母变体。

脚本 18-7 不必在书中查找加重音的字符，可以让 JavaScript 在需要时生成列表

```
javascript:(function(){var eWin,n,i,j,w,t='<table border=1 cellpadding=20 cellspacing=0>', l=document.
→createElement('p'),v='aAeEiIoOuUyY',s=new Array('acute','circ','elig','Elig','grave','ring',
→'slash', 'tilde','uml');for(i=0;i<v.length; i++){for(j=0;j<s.length;j++){w=v.charAt(i)+s[j]+';';
→l.innerHTML='&'+w;n=l.innerHTML;if(n.length==1){t+='<tr><td>&'+w+'</td><td>&'+'amp;'+w+'</td>
→<td>&'+'amp;#'+n.charCodeAt(0)+';</td></tr>';}}}eWin=window.open('','EW','scrollbars=yes,width=300,
→height='+screen.height);eWin.document.body.innerHTML=t+'</table>';})();
```

⇛ 显示 ISO Latin 字符

```
1. var eWin,n,i,j,w,t = '<table border=1 cellpadding=20 cellspacing=0>', l = document.create
  →Element('p'),
```

这个 bookmarklet 首先对 4 个变量进行初始化。

```
2. v = 'aAeEiIoOuUyY',
```

对字符串 v 进行初始化，其中包含所有元音字母。

```
3. s = new Array('acute', 'circ', 'elig', 'Elig', 'grave', 'ring', 'slash', 'tilde', 'uml');
```

这是一个数组 s，其中包含所有变音字符编码。

```
4. for (i=0;i<v.length;i++) {
```

这一行用 i 循环遍历 v 字符串中的字符。

```
5. for (j=0;j<s.length;j++) {
```

这一行用 j 遍历 s 数组。

```
6. w = v.charAt(i) + s[j] + ';';
```

在这一行中，将变量 w 设置为元音字母加上编码，后接分号。

```
7. l.innerHTML = '&' + w;
   n = l.innerHTML;
```

对于下一个步骤，我们不需要实体的字符串表示，而是需要实体本身——也就是说，需要 ã 而不是ã。可以获取前一步中设置的字符串 w，在前面加上&符号，然后放在已经创建的元素 l 的 innerHTML 中。这会把实体的字符串值转换为显示的值。为了使用这个值，把变量 n 设置为同一个 innerHTML 的内容。

```
8. if (n.length==1) {
```

为了获得所有可能的实体，我们的元音和变音字符列表包含一些无效的组合。如果前面的转换步骤实际上没有转换我们的字符串，就说明这是无效的组合。例如，æ转换为 æ，Á转换为 Á；但是，&Aelig;不是有效的实体（尽管它看起来应该转换为 Æ），所以前一步不会转换它，结果仍然是一个 7 字符的字符串。有效的实体最终会产生一个 1 字符的字符串，并执行后面的代码。

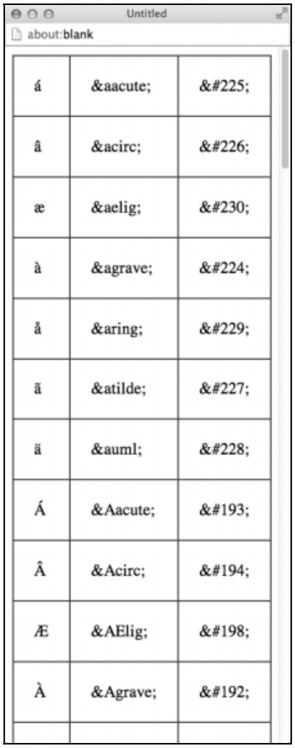
```
9. t += '<tr><td>&' + w + '</td><td>&' + 'amp;'+ w + '</td><td>&' + 'amp;#' + n.charCodeAt(0) + ';
  →</td></tr>';
```

我们已经获得了一个有效的实体，现在要把它放到表格中。表格的第一列显示实体本身，第二列

显示字符串表示，第三列显示数字表示。最后一个值是通过在实体上使用 `charCodeAt()`方法获得的。

10.
```
eWin = window.open('','EW','scrollbars=yes,width=300,height=' + screen.height);
eWin.document.body.innerHTML = t + '</table>';
```

完成之后，打开一个新窗口，然后把表格写到窗口中，见图 18-18。



á	á	á
â	â	â
æ	æ	æ
à	à	à
å	å	å
ã	ã	ã
ä	ä	ä
Á	Á	Á
Â	Â	Â
Æ	Æ	Æ
À	À	À

图 18-18　这个脚本的结果是一个以 HTML 实体形式显示元音的所有变体的新窗口

✓提示

- ❑ 这个示例的表格很长，所以在处理打开窗口的高度时，使用了一个小技巧。并不为窗口指定固定的高度，而是根据用户的显示器高度设置窗口高度。如果你不希望这样，可以把它设置为固定的大小。
- ❑ Æ 的 HTML 实体是`Æ`——也就是说，需要大写的 E，而不是小写的 e。现在你可以体会到这样的 bookmarklet 是多么方便了。

18.7　将 RGB 值转换为十六进制

另一个对于 Web 开发人员有帮助的小工具是 RGB 到十六进制的转换器。当需要将来自图形程序（比如 Adobe Photoshop 或者 Fireworks）的颜色值转换为浏览器颜色值时，以用作页面背景或文本的颜色，这会非常有用。脚本 18-8 演示如何用 JavaScript 编写转换计算器并且将代码转换为 bookmarklet。

脚本 18-8 这个脚本接受 RGB 颜色值，并且将它们转换为等同的十六进制值

```
javascript:(function(){var s,i,n,h='#',x='0123456789ABCDEF',c=prompt('R,G,B:','');if(c){s=c.split
➝(',');for(i=0;i<3;i++){n=parseInt(s[i]);h+=x.charAt(n>>4)+x.charAt(n&15);}prompt('Hexcolor:',h);
➝}})();
```

⇛ 将 RGB 值转换为十六进制

1. `var s,i,n,h = '#',`

这个 bookmarklet 首先对 4 个变量进行初始化。

2. `x = '0123456789ABCDEF',`

变量 x 被设置为有效的十六进制值。

3. `c = prompt('R,G,B:','');`

这一行提示用户输入所需的 RGB 值（以逗号分隔），如图 18-19 所示。

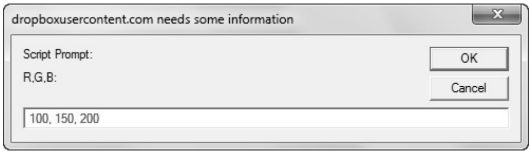


图 18-19 这个脚本的第一部分提示用户输入 RGB 值

4. `if (c) {`

如果用户输入了任何内容，就继续执行代码。否则，c 的值就是 null，bookmarklet 跳过后面所有的步骤。

5. `s = c.split(',');`

用逗号分隔 c 中的内容，并且将结果放进 s 数组中。

6. `for (i=0;i<3;i++){`

对于红色、绿色和蓝色值，循环执行以下代码。

7. `n = parseInt(s[i]);`

将 s 的当前元素转换为数字，并且将它保存为 n。

8. `h += x.charAt(n>>4) + x.charAt(n&15);`

这一行将 n 转换为两个十六进制数字，并且将结果添加到 h 中。

9. `prompt('Hexcolor:',h);`

通过提示命令显示结果（准备将它复制进 HTML 页面），如图 18-20 所示。采用这种做法而不是用警告对话框，这样就可以以后复制并且粘贴代码。

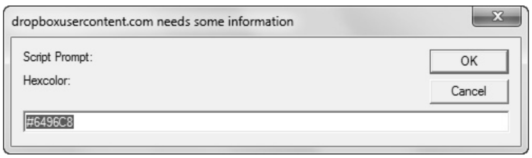


图 18-20 另一个提示框提供了计算出的十六进制值

18.8　对值进行转换

值可以从一种类型转换成其他各种可能的类型。脚本 18-9 仅仅演示一个示例：如何将公里数转换为英里数。

脚本 18-9　可以创建 bookmarklet 来执行几乎任何类型的单位转换。这个脚本将公里数转换为英里数

```
javascript:(function(){var t,expr=prompt('Length in kilometers:','');if(isNaN(parseFloat(expr)))
→{t=expr+' is not a number';}else{t='Length in miles is '+Math.round(expr*6214)/10000;}alert(t);})();
```

⇒ 将公里数转换为英里数

1. `var t,expr = prompt('Length in kilometers:','');`

这个 bookmarklet 首先提示用户输入以公里为单位的长度（见图 18-21）。

2. `if (isNaN(parseFloat(expr))) {`

检查用户是否输入了数字值。

3. `t = expr + ' is not a number';`

如果没有输入数字值，则弹出错误消息。

4. else{

```
    t = 'Length in miles is' + Math.round(expr*6214)/10000;
  }
```

否则，将值转换为英里数并且将它存储在 t 中。

5. `alert(t);`

无论输入值如何，我们将转换结果存储在 t 中，结果显示如图 18-22 所示。



图 18-21　首先，要求用户输入要转换的数字

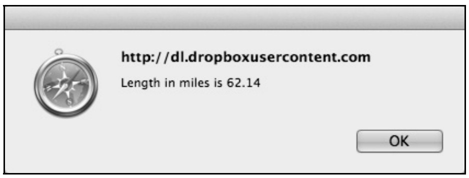


图 18-22　JavaScript 返回转换结果

✓提示

- 改写这个脚本来执行所需的任何转换都是很简明的。只需改变步骤 1 中的标签，并且将步骤 4 中的数学表达式替换为所需的转换的适当表达式。
- 可以建立一批执行不同转换的 bookmarklet，将它们组织在 Bookmarks 或 Favorites 菜单中的文件夹中。然后就可以通过鼠标单击进行转换。

18.9　bookmarklet 计算器

如果你考虑一下，会觉得只用一长行代码建立一个完全成熟的计算器是非常困难的。但是，可以使用脚本 18-10 这样的 bookmarklet，利用 JavaScript 内置的 Math 函数来执行相当复杂的计算。

脚本 18-10 可以用这个 bookmarklet 进行相当复杂的计算

```
javascript:(function(){var evl,expr=prompt('Formula...(eg: 2*3 + 7/8)','');with(Math)try{evl=parseFloat
→(eval(expr));if(isNaN(evl)){throw Error('Not a number!');}prompt('Result of '+expr+':',evl);}
→catch(evl){alert(evl);}})();
```

⇛ 使用 JavaScript 计算器

1. `var evl,expr = prompt('Formula...(eg: 2*3 + 7/8)','');`

这一行设置了一个变量 `evl`，然后提示用户输入一个表达式或公式（如图 18-23 所示），将它存储在 `expr` 中。

2. `with(Math) try {`

下面几行需要使用 JavaScript 内置的 Math 例程进行运算。`with(Math)`这一行告诉解释器，当看到这些函数时，应该将它们作为 Math 命令对待。

`try{}`警告 JavaScript，我们将要做的事情可能会失败，如果出现问题，不要惊慌。实际上，如果出现问题，JavaScript 甚至不必弹出错误消息，因为我们要自己处理错误。

3. `evl = parseFloat(eval(expr));`

计算表达式，将结果转换为浮点数，然后存储在 `evl` 中。

4. `if (isNaN(evl)) {`

如果 `evl` 中的值不是数字（`NaN`），那么执行下一行。

5. `throw Error('Not a number!');`

如果执行到这里，就说明由于某种原因，用户输入的内容无法计算出数字。当发生这种情况时，我们希望显示错误消息“Not a number!”。这里设置这个消息，它将在步骤 7 中显示。

6. `prompt('Result of ' + expr + ':', evl);`

否则，表达式就是有效的，因此显示计算结果，如图 18-24 所示。

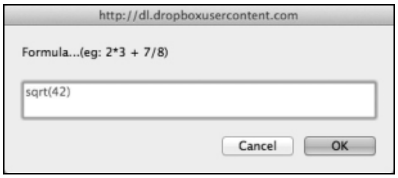


图 18-23 提示用户必须输入一个公式

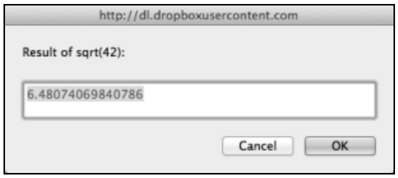


图 18-24 JavaScript 返回计算结果

7.
```
catch(evl) {
   alert(evl);
 }
```

这里是步骤 2 开始的 `try{}`块的末尾。如果到达这里，就是发生了两种情况之一：要么是在步骤 5 中遇到了错误，或者是发生了其他错误。无论是哪种情况，我们都“捕获抛出的错误”，并且将它放在屏幕上的警告框中。

✔提示

- ❑ 还记得 `try/throw/catch` 语法的意义吗？我们在 2.11 节中最先讨论过这些语法。
- ❑ 如果你想知道可以使用哪些 Math 函数，回头看看第 3 章，我们在 3.1 节介绍了 Math 对象及其方法。

18.10　缩短 URL

由于许多原因，我们可能希望获得更短的 URL 版本，比如你可能正在使用与 Twitter 相似的服务，它对字符数量有限制，或者要把结果粘贴到电子邮件中而且不希望它换行。无论出于什么原因，脚本 18-11 都可以提供更简单的 URL。

脚本 18-11　只需单击一次鼠标（和这个脚本），就可以缩短这些 URL

```
javascript:(function(){window.open('http://tinyurl.com/create.php?url='+location.href,'','width=750,
➝height=500,scrollbars=yes');})();
```



图 18-25　许多地方需要短 URL 版本，这个 bookmarklet 可以很方便地提供短 URL

⇒ 缩短 URL

- ❑ `window.open('http://tinyurl.com/create.php?url=' + location.href,'','width=750,`
  `➝height=500,scrollbars=yes');`

这里打开一个新窗口并使用 TinyURL.com 服务设置短 URL。我们向这个服务传递当前页面位置（location.href），这就是它需要的所有信息。TinyURL.com 会立即把缩短的 URL 放到你的剪贴板中，所以只需访问正确的网页，单击这个 bookmarklet，看看打开的页面是否一切正常（见图 18-25），关闭它，然后把新的 URL 粘贴到所需的任何地方。

✓提示

- ❑ 网上有许多不同的 URL 缩短服务。如果你不喜欢 TinyURL.com，可以试试 bit.ly、goo.gl 或 is.gd。
- ❑ 如果你也在 Twitter 上，请和我们打个招呼，我们的用户名是@negrino 和@dori。

18.11　检验页面

在创建页面时，最好确保页面符合 Web 标准。这样的页面在现代浏览器中装载得更快，而且更容易维护。对页面进行代码有效性检查的最容易的方法是，通过 W3C 维护的页面检验器（http://validator.w3.org）运行它。脚本 18-12 所示的 bookmarklet，检查当前在浏览器中显示的页面的有效性。实现方法是，获得当前页面的 URL，将它传递给检验器，然后打开一个显示检验器结果的新窗口，如图 18-26 所示。

脚本 18-12　使用这个脚本确保页面包含符合 Web 标准的有效的标记

```
javascript:(function(){window.open('http://validator.w3.org/check?uri='+location.href,'',
→'width=800,height=900,resizable=yes,scrollbars=yes');})();
```

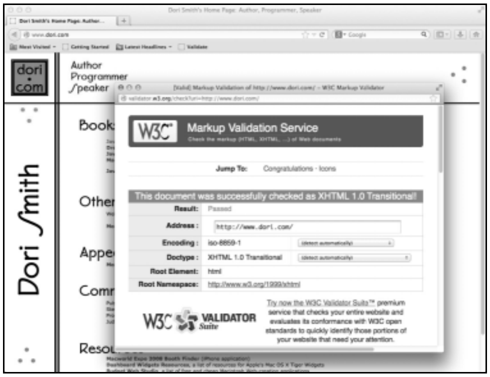


图 18-26　通过 W3C 检验器运行你的页面，如果页面编写得正确，就会返回让人满意的结果

⇛ 检验页面

```
❑ window.open('http://validator.w3.org/check?uri=' + location.href,'','width=800,height=900,
  →resizable=yes,scrollbars=yes');
```

这个 bookmarklet 很有用，而且并不复杂。首先，打开一个窗口，并且将检验器的 URL 传递给这个窗口。你会注意到，检验器有一个参数 `uri`，它接受当前页面的 URL，即 `location.href`。两者之间的加号将位置对象与检验器的 URL 拼接起来。这一行的其余部分仅仅是窗口大小和其他属性的一些参数。

18.12　通过电子邮件发送页面

在网上冲浪时，有时候会发现一个页面非常有用，需要与同事分享，或者很有趣，希望与好朋友分享。这个 bookmarklet（见脚本 18-13）获取用户当前所处的页面，加上突出显示的文本，使用这些内容创建一个新的电子邮件。

脚本 18-13　如果希望通过电子邮件把一个网页的所有内容或一部分内容发送给别人，使用这个 bookmarklet 是最简单的方法

```
javascript:(function(){location.href='mailto:?SUBJECT='+document.title+'&BODY='+escape(location.href)+
→' \r'+window.getSelection();})();
```

⇛ 通过电子邮件发送网页

```
❑ location.href = 'mailto:?SUBJECT=' + document.title + '&BODY=' + escape(location.href) +
  →'\r' + window.getSelection();
```

如果你以前在网页上添加过 `mailto` 链接，就应该认识这种语法。这行代码的作用相当于单击`<a href='mailto:'></a>`链接。电子邮件的主题设置为当前文档的标题，邮件体设置为页面的 URL 和当前选择的文本，见图 18-27。



图 18-27　只需一次单击就会打开一个新窗口，其中已经设置了邮件的基本元素

✓提示

- ❑ 对于这个 bookmarklet，不必非使用 Gmail 或 Web mail。只要计算机上设置了默认的邮件客户端，它就会打开并创建邮件。

18.13　改变页面大小

在开发网站时，能够看看页面在更小的显示屏幕上的效果很有用。这个 bookmarklet（见脚本 18-14）把浏览器窗口修改为 640×480。

脚本 18-14　如果希望看看页面在采用另一种大小时效果如何，使用这样的 bookmarklet 会很方便

```
javascript:(function(){resizeTo(640,480);moveTo(0,0);})();
```

⇛ 改变页面大小

- ❑ `resizeTo(640,480);moveTo(0,0);`

第一个命令 `resizeTo()`修改浏览器窗口的尺寸。下一个命令 `moveTo()`告诉浏览器左上角位于何处（见图 18-28）。



图 18-28　一眼就可以看出这个站点的小屏幕版本显示不了大量内容

✔提示

- 这个 bookmarklet 本身用处不大。但是，可以创建一系列与它几乎完全相同的 bookmarklet，分别针对你希望检查的所有窗口大小。显然，针对任何窗口大小修改这个 bookmarklet 都是非常简单的。
- 以相同大小并列显示多个窗口也是有用的（如果显示屏幕足够大）。例如，可以通过一个 bookmarklet 把窗口宽度设置为 700 像素，高度设置为显示屏幕的最大高度：

  ```
  resizeTo(700,screen.availHeight);
  moveTo(0,0);
  ```

  再创建另一个 bookmarklet：

  ```
  resizeTo(700,screen.availHeight);
  moveTo(screen.availWidth-700,0);
  ```

  后者也把屏幕宽度设置为 700 像素，高度设置为显示屏幕的最大高度，但是把窗口定位在屏幕的右边而不是左边。如果显示屏幕是 1400 像素宽或更宽，就会看到它们并排显示，没有重叠。
- 如果你设置的 bookmarklet 不起作用，检查一下你调整的窗口是否只打开了一个标签页。

附录 A

JavaScript的版本演化和参考资料

自从作为 Netscape Navigator 2.0 的一部分出现以来，JavaScript 经过了 20 来年的演化。这个附录简要地讨论了 JavaScript 的不同版本，以及各种浏览器所包含的 JavaScript 版本。

这里还提供了一个 JavaScript 对象关系图以及一个对象表，其中列出了大多数 JavaScript 软件对象，以及它们的属性、方法和事件处理程序，并且包括 ECMAScript 3。

A.1　JavaScript版本

JavaScript 有几个不同的名称（具体视你所用的产品而定），而且有十几个不同的版本。除了 JavaScript 之外，还有 JScript 和 ECMAScript。下面分别介绍各个版本。

A.1.1　Netscape的JavaScript

JavaScript 的第一个版本最初称为 LiveScript，它是在 Netscape Navigator 2.0 中首次发布的。Netscape 希望 LiveScript 成为一种扩展浏览器功能的方式，并且让 Web 设计人员能够给他们的站点增加交互性。Navigator 2.0 中的 JavaScript 版本是 JavaScript 1.0。

Navigator 3.0 中提供了 JavaScript 1.1，其中增加了对图像、数组、Java applet 和插件的支持，还做了许多其他修改。

随着 Navigator 4.0（也称为 Netscape Communicator）的发布，JavaScript 1.2 出现了，它做了更多的改进和调整。Netscape 4.5 随后提供了 JavaScript 1.3。JavaScript 1.4 是纯服务器端语言，而 Netscape 6 引入了 JavaScript 1.5。

JavaScript 的当前版本是由开源的 Mozilla 项目开发的，主要考虑的是它的 Firefox 浏览器。Firefox 较新的版本支持 ECMA-Script-262 Edition 5（见下面的讨论）。

到编写本书时，JavaScript 的当前版本是 1.8.6，随 Firefox 17 一起发布（见表 A-1）。

表A-1　JavaScript版本

浏　览　器	JavaScript版本
Netscape 2	1.0
Netscape 3	1.1
Netscape 4.0~4.05	1.2
Netscape 4.06~4.7	1.3
Netscape 6.0、Netscape 7.0、Mozilla、Firefox 1.0~1.41	1.5
Firefox 1.5	1.6
Firefox 2	1.7
Firefox 3	1.8
Firefox 4	1.8.5
Firefox 17+	1.8.6

A.1.2　微软的JScript

微软按照自己的方式实现 JavaScript，这个版本并不总是与 Netscape 版本兼容。JavaScript 的微软版本称为 JScript 1，它或多或少与 JavaScript 1.0 兼容，但是有一些差异。很自然，JScript 只出现在 Windows 和微软的 IE（MSIE）版本中。

在 Windows 上，还有用于 Windows 95/NT 的 JScript 2（多少可与 JavaScript 1.1 比较），MSIE 3.02 的升级版本和以后的版本支持 JScript 2。并非 MSIE 3.02 的所有版本都支持 JScript 2.0。要想查明已经安装的 JScript 是哪个版本，可以在硬盘上搜索 jscript.dll。查看这个文件的属性，并且单击 Version 选项卡。如果文件版本不是至少以 2 开头的，那么浏览器早就应该升级了。

在 Macintosh 上，MSIE 3.0 没有 JScript，但是 3.01 中有。它包含 JScript 1.0，但是与 Windows 上的版本不同。在 JScript 的 Mac 和 Windows 版本之间有一些差异（例如，Mac 版本支持 Image 对象上的鼠标翻转器，而 Windows JScript 1.0 不支持）。在 2003 年，微软放弃了 Mac 的 MSIE，并且于 2005 年停止了支持。

觉得乱吗？别忙，还有呢：JScript 3.0 大致相当于 JavaScript 1.2，JScript 5.x 大致相当于 JavaScript 1.5。根据 IE 的版本和所运行的 Windows，表 A-2 有助于你了解 JScript 的版本。

表A-2　JScript版本

浏　览　器	JavaScript版本
MSIE 4	3.0
MSIE 5	5.0
MSIE 5.5	5.5
MSIE 6	5.6
MSIE 7	5.7
MSIE 8	5.8
MSIE 9+	9.0

A.2　ECMAScript

在 1996 年，Web 开发人员发现，Netscape 沿着一个方向发展 JavaScript，而微软沿着一个在一定程度上兼容但有差异的方向发展 JScript。没有人喜欢这种分裂局面，Web 开发人员不得不在页面中处理 JavaScript 的不同"方言"，否则他们的代码就只能在一种浏览器中工作。开发人员希望有个标准。所以 Netscape 加入了一个国际标准组织 ECMA，并且向它提交了 JavaScript 语言规范，微软也拿出了自己的意见和建议。ECMA 开始制定 JavaScript 标准，并且于 1997 年 6 月形成了 ECMA-262 标准（也称为 ECMAScript，这个术语一般只在闲谈中使用）。这个标准与 JavaScript 1.1 非常相似，但不完全一样，后续版本纠正了这个问题。如果你想阅读正式的 ECMAScript 规范，可以从 www.ecma-international.org/ 下载。找到 Standards 链接，然后通过它找到 ECMA-262 规范。

自从 1997 年以来，ECMAScript 经历了几种版本：最重要的是，第 3 版在 1999 年 12 月发布，第 5 版在 2009 年 12 月发布，5.1 版在 2011 年 6 月发布。第 4 版一直没有发布，最终取消了。现在的浏览器支持第 3 版（约与 JavaScript 1.5 相当），并且慢慢兼容第 5 版。一定要注意的是，ECMAScript 现在控制着 JavaScript 标准的发展方向。当前所有浏览器厂商都让自己的 JavaScript 实现与 ECMAScript 兼容。

所以，只要你编写的代码符合 ECMAScript 标准，它就应该能够在 MSIE 4 和 Netscape Navigator 6 中正常运行。但是，仍然应该在不同的浏览器、平台和版本中测试你的代码。

基于 WebKit 的苹果 Safari 和基于 Blink 的谷歌 Chrome 一直支持 ECMAScript。

A.3　对象表

JavaScript 书都应该提供 JavaScript 对象表，包括它们的属性、方法和事件处理程序（这些术语的定义见第 1 章）。表 A- 3 列出了大多数 JavaScript 对象并且包括 ECMAScript 3。我们省略了几个很不常用的对象，以及较早版本中的一些老式对象，这些对象已经被新对象或扩展对象取代了。

表A-3　JavaScript对象表

对　象	属　性	方　法	事件处理程序
Anchor	name	无	无
anchors数组	length	无	无
Applet	align	applet方法	onblur
	code	blur	onclick
	codeBase	focus	ondblclick
	height		onfocus
	hspace		onkeydown
	name		onkeypress
	vspace		onkeyup
	width		onmousedown
			onmousemove
			onmouseout
			onmouseover

（续）

对　　象	属　　性	方　　法	事件处理程序
			onmouseup
			onresize
applets数组	length	无	无
Area	alt	无	onblur
	cords		onclick
	hash		ondblclick
	host		onfocus
	hostname		onkeydown
	href		Onkeypress
	noHref		onkeyup
	pathname		onmousedown
	port		onmousemove
	protocol		onmouseout
	search		onmouseover
	shape		onmouseup
	target		onresiz
Array	length	concat	无
		join	
		reverse	
		slice	
		sort	
		splice	
		toLocaleString	
		toString	
Body	alink	无	onblur
	Background		onclick
	bgColor		ondblclick
	link		onfocus
	text		onkeydown
	vlink		onkeypress
			onkeyup
			onmousedown
			onmousemove
			onmouseout
			onmouseover
			onmouseup
			onresize
Button	form	blur	onblur
	name	click	onchange
	type	focus	onclick
	value		ondblclick

（续）

对 象	属 性	方 法	事件处理程序
			onfocus onkeydown onkeypress onkeyup onmousedown onmousemove onmouseout onmouseover onmouseup
Checkbox	checked defaultChecked form name type value	blur click focus	onblur onchange onclick ondblclick onfocus onkeydown onkeypress onkeyup onmouseddown onmousemove onmouseout onmouseover onmouseup
Date	无	getDate getDay getFullYear getHours getMilliseconds getMinutes getMonth getSeconds getTime getTimezoneOffset getUTCDate getUTCDay getUTCFullYear getUTCHours getUTCMilliseconds getUTCMinutes getUTCMonth getUTCSeconds getYear parse	无

（续）

对　　象	属　　性	方　　法	事件处理程序
		setDate	
		setFullYear	
		setHours	
		setMilliseconds	
		setMinutes	
		setMonth	
		setSeconds	
		setTime	
		setUTCDate	
		setUTCFullYear	
		setUTCHours	
		setUTCMilliseconds	
		setUTCMinutes	
		setUTCMonth	
		setUTCSeconds	
		setYear	
		toGMTString	
		toLocaleDateString	
		toLocaleString	
		toLocaleTimeString	
		toString	
		toUTCString	
		UTC	
		valueOf	
document	alinkColor	clear	onblur
	Anchor	close	onclick
	anchors	createElement	ondblclick
	Applet	createTextNode	onfocus
	applets	focus	onkeydown
	Area	getElementById	onkeypress
	attributes	getElementsByName	onkeyup
	bgColor	getElementsByTagName	onmousedown
	Body	Open	onmousemove
	childNodes	Write	onmouseout
	cookie	writeIn	onmouseover
	documentElement		onmouseup
	domain		onresize
	embed		
	embeds		
	fgColor		
	firsrChild		
	Form		

（续）

对　象	属　性	方　法	事件处理程序
	forms		
	lmage		
	images		
	lastChild		
	lastModified		
	linkColor		
	Link		
	links		
	location		
	namespaceURI		
	nextSibling		
	nodeName		
	nodeType		
	parentNode		
	plugins		
	previousSibling		
	referrer		
	Script		
	StyleSheet		
	styleSheets		
	title		
	URL		
	vlinkColor		
FileUpload	form	blur	onblur
	name	focus	onclick
	type	select	ondblclick
	value		onfocus
			onkeydown
			onkeypress
			onkeyup
			onmousedown
			onmousemove
			onmouseout
			onmouseover
			onmouseup
			onselect
Form	action	reset	onclick
	Button	submit	ondblclick
	Checkbox		onkeydown
	elements		onkeypress
	encoding		onkeyup
	encType		onmousedown

（续）

对　象	属　性	方　法	事件处理程序
	FileUpload		onmousemove
	Hidden		onmouseout
	length		onmouseover
	method		onmouseup
	name		onreset
	Password		onsubmit
	Radio		
	Reset		
	Select		
	Submit		
	target		
	Text		
	Textarea		
forms array	length	无	无
hidden	form	无	无
	maxLength		
	name		
	readOnly		
	size		
	type		
	value		
History	length	back	无
		forward	
		go	
history数组	length	无	无
Image	align	无	onabort
	alt		onblur
	border		onclick
	complete		ondblclick
	height		onerror
	href		onkeydown
	hspace		onkeypress
	isMap		onkeyup
	lowsrc		onload
	name		onmousedown
	src		onmousemove
	useMap		onmouseout
	vspace		onmouseover
	width		onmouseup
	x		onreset
	y		onresize
			onsubmit

（续）

对　　象	属　　性	方　　法	事件处理程序
images数组	length	无	无
Link	hash	无	onblur
	host		onclick
	hostname		ondblclick
	href		onkeydown
	name		onkeypress
	pathname		onkeyup
	port		onmousedown
	protocol		onmousemove
	rel		onmouseout
	rev		onmouseover
	search		onmouseup
	target		
links数组	length	无	无
location	hash	assign	无
	host	reload	
	hostname	replace	
	href		
	pathname		
	port		
	protocol		
	search		
Math	E	abs	无
	LN2	acos	
	LN10	asin	
	LOG2E	atan	
	LOG10E	atan2	
	PI	ceil	
	SQRT1_2	cos	
	SQRT2	exp	
		floor	
		log	
		max	
		min	
		pow	
		random	
		round	
		sin	
		sqrt	
		tan	

（续）

对　　象	属　　性	方　　法	事件处理程序
MimeType	description	无	无
	enabledPlugin		
	suffixes		
	type		
mimeTypes数组	length	无	无
navigator	appCodeName	javaEnabled	无
	appName	taintEnabled	
	appVersion		
	cookieEnabled		
	MimeType		
	mimeTypes		
	platform		
	Plugin		
	plugins		
	userAgent		
Number	MAX_VALUE	toExponential	无
	MIN_VALUE	toFixed	
	NaN	toLocaleString	
	NEGATIVE_INFINITY	toPrecision	
	POSITIVE_INFINITY	toString	
		valueOf	
Object	attributes	appendChild	onblur
	childNodes	blur	onchange
	children	click	onclick
	className	cloneNode	oncontextmenu
	clientHeight	focus	ondblclick
	clientLeft	getAttribute	onfocus
	clientTop	getAttributeNode	onkeydown
	clientWidth	getElementsByTagName	onkeypress
	dir	getExpression	onkeyup
	firstChild	hasChildNodes	onmousedown
	id	insertBefore	onmousemove
	innerHTML	item	onmouseout
	lang	releaseCapture	onmouseover
	language	removeAttribute	onmouseup
	lastChild	removeAttributeNode	onreadystatechange
	length	removeChild	onresize
	localName	replaceChild	onscroll
	namespaceURI	scrollIntoView	
	nextSibling	setAttribute	
	nodeName		

（续）

对 象	属 性	方 法	事件处理程序
	nodeType		
	nodeValue		
	offsetHeight		
	offsetLeft		
	offsetParent		
	offsetTop		
	offsetWidth		
	ownerDocument		
	parentNode		
	prefix		
	previousSibling		
	readyState		
	scrollHeight		
	scrollLeft		
	scrollTop		
	scrollWidth		
	sourceIndex		
	style		
	tabIndex		
	tagName		
	title		
Option	defaultSelected	remove	无
	form		
	index		
	selected		
	text		
	value		
Options数组	length	无	无
Password	defaultValue	blur	onblur
	form	focus	onchange
	maxLength	select	onclick
	name		ondblclick
	readOnly		onfocus
	size		onkeydown
	type		onkeypress
	value		onkeyup
			onmousedown
			onmousemove
			onmouseout
			onmouseover
			onmouseup
			onselect

（续）

对　　象	属　　性	方　　法	事件处理程序
Plugin	description	refresh	无
	filename		
	length		
	name		
plugins数组	length	无	无
Radio	checked	blur	onblur
	defaultChecked	click	onchange
	form	focus	onclick
	name		ondblclick
	type		onfocus
	value		onkeydown
			onkeypress
			onkeyup
			onmousedown
			onmousemove
			onmouseout
			onmouseover
			onmouseup
radio数组	length	无	无
RegExp	constructor	compile	无
	global	exec	
	ignoreCase	test	
	input	toSource	
	lastIndex	toString	
	lastMatch	valueOf	
	lastParen		
	leftContext		
	multiline		
	rightContext		
	source		
	$1		
	$2		
	$3		
	$4		
	$5		
	$6		
	$7		
	$8		
	$9		
	$_		
	$*		

（续）

对　象	属　性	方　法	事件处理程序
	$&		
	$+		
	$`		
	$'		
Reset	form	blur	onblur
	name	click	onclick
	type	focus	ondblclick
	value		onfocus
			onkeydown
			onkeypress
			onkeyup
			onmousedown
			onmousemove
			onmouseout
			onmouseover
			onmouseup
screen	availHeight	无	无
	availWidth		
	colorDepth		
	height		
	pixelDepth		
	width		
Script	defer	无	
	event		
	htmlFor		
	language		
	src		
	text		
	type		
Select	form	blur	onblur
	length	focus	onchange
	multiple		onclick
	name		ondblclick
	Option		onfocus
	options		onkeydown
	selectedIndex		onkeypress
	size		onkeyup
	type		onmousedown
	value		onmousemove
			onmouseout
			onmouseover
			onmouseup
			onresize

（续）

对　　象	属　　性	方　　法	事件处理程序
String	length	anchor	无
		big	
		blink	
		bold	
		charAt	
		charCodeAt	
		concat	
		fixed	
		fontcolor	
		fontsize	
		fromCharCode	
		indexOf	
		italics	
		lastIndexOf	
		link	
		localeCompare	
		match	
		replace	
		search	
		slice	
		small	
		split	
		strike	
		sub	
		substr	
		substring	
		sup	
		toLocaleLowerCase	
		toLocaleUpperCase	
		toLowerCase	
		toString	
		toUpperCase	
		valueOf	
Style	background	无	无
	backgroundAttachment		
	backgroundColor		
	backgroundImage		
	backgroundPosition		
	backgroundRepeat		
	border		
	borderBottom		
	borderBottomColor		

（续）

对 象	属 性	方 法	事件处理程序
	borderBottomStyle		
	borderBottomWidth		
	borderColor		
	borderLeft		
	borderLeftColor		
	borderLeftStyle		
	borderLeftWidth		
	borderRight		
	borderRightColor		
	borderRightStyle		
	borderRightWidth		
	borderStyle		
	borderTop		
	borderTopColor		
	borderTopStyle		
	borderTopWidth		
	borderWidth		
	bottom		
	clear		
	clip		
	color		
	cssText		
	cursor		
	direction		
	display		
	font		
	fontFamily		
	fontSize		
	fontStyle		
	fontVariant		
	fontWeight		
	height		
	left		
	lineHeight		
	listStyle		
	listStyleImage		
	listStylePosition		
	listStyleType		
	margin		
	marginBottom		
	marginLeft		
	marginRight		

（续）

对　象	属　性	方　法	事件处理程序
	marginTop		
	overflow		
	padding		
	paddingBottom		
	paddingLeft		
	paddingRight		
	paddingTop		
	pageBreakAfter		
	pageBreakBefore		
	position		
	right		
	tableLayout		
	textAlign		
	textIndent		
	textTransform		
	top		
	unicodeBidi		
	verticalAlign		
	visibility		
	whiteSpace		
	width		
	wordSpacing		
	zIndex		
StyleSheet	cssRules	无	无
	disabled		
	href		
	media		
	parentStyleSheet		
	title		
	type		
Submit	form	blur	onblur
	name	click	onclick
	type	focus	ondblclick
	value		onfocus
			onkeydown
			onkeypress
			onkeyup
			onmousedown
			onmousemove
			onmouseout
			onmouseover
			onmouseup

（续）

对　　象	属　　性	方　　法	事件处理程序
Text	defaultValue	blur	onblur
	disabled	click	onchange
	form	focus	onclick
	maxLength	select	ondblclick
	name		onfocus
	readOnly		onkeydown
	size		onkeypress
	type		onkeyup
	value		onmousedown
			onmousemove
			onmouseout
			onmouseover
			onmouseup
			onselect
Textarea	cols	blur	onblur
	defaultValue	click	onchange
	form	focus	onclick
	name	select	ondblclick
	readOnly		onfocus
	rows		onkeydown
	type		onkeypress
	value		onkeyup
	wrap		onmousedown
			onmousemove
			onmouseout
			onmouseover
			onmouseup
			onselect
window	closed	alert	onblur
	defaultStatus	blur	onerror
	document	clearInterval	onfocus
	event	clearTimeout	onload
	external	close	onmove
	frames	confirm	onresize
	history	focus	onunload
	length	moveBy	
	location	moveTo	
	name	open	
	navigator	print	
	opener	prompt	
	parent	resizeBy	
	screen	resizeTo	

（续）

对　　象	属　　性	方　　法	事件处理程序
	self	scroll	
	status	scrollBy	
	top	scrollTo	
		setInterval	
		setTimeout	
XMLHttpRequest	readyState	abort	onload
	response	addEventListener	onloadend
	responseText	getAllResponseHeaders	onloadstart
	responseXML	getResponseHeader	onreadystatechange
	status	open	ontimeout
	statusText	overrideMimeType	
	timeout	send	
	upload	setRequestHeader	
	withCredentials		

如果你需要在自己的浏览器中显示所有的对象，检查一下相应浏览器的开发者工具。例如，图 A-1 显示了在 Chrome 开发者工具中查看，尽管并非所有内容都是跨浏览器有效的，但可以作为学习的起点。



图 A-1　使用 Chrome Developers Tools，你可以清楚地了解浏览器如何查看代码。其他浏览器也有类似功能的开发者工具

附录 B

JavaScript保留字

保留字是对 JavaScript 有特殊含义的单词。因此，不能将它们用作变量名或函数名。

在前面的章节中，你已经见过了许多保留字，但是对其他保留字可能还不熟悉。其中一部分是预留的保留字，也就是说，它们可能是 ECMAScript 未来版本中的命令。现在就应该避免使用它们，以免在新版本发布时不得不修改代码。

ECMAScript 3 保留字

从 ECMAScript 3 开始，这些单词是 JavaScript 语言的一部分。

```
break       for          throw
case        function     try
catch       if           typeof
continue    in           var
default     instanceof   void
delete      new          while
do          return       with
else        switch
finally     this
```

ECMAScript 3 未来的保留字

```
abstract    final        protected
boolean     float        public
byte        goto         short
char        implements   static
class       import       super
const       int          synchronized
debugger    interface    throws
double      long         transient
enum        native       volatile
export      package
extends     private
```

ECMAScript 5 未来的保留字

这里 ES5 的保留字列表。它与 ES3 类似，考虑到浏览器主要支持 ES3，我们建议你记住所有列表。

break	finally	this
case	for	throw
catch	function	try
continue	if	typeof
debugger	in	var
default	instanceof	void
delete	new	while
do	return	with
else	switch	

ES5 未来的保留字

class	interface	super
const	let	yield
enum	package	
export	private	
extends	protected	
implements	public	
import	static	

应该避免使用的其他标识符

附录 A 中列出的对象名不是正式的保留字（不属于上面列出的关键字），但是因为它们是 JavaScript 语言的一部分，你不应该将它们用作变量名或函数名。如果这么做，结果无法预料。

另外，大多数浏览器是区分大小写的，这意味着 Document 和 document 是不同的。IE 只在某些时候是区分大小写的，这意味着它可能无法区分 Document 和 document。因此，即使代码能够在一种浏览器中正常工作，也不意味着在其他浏览器中也能正常工作。一定要进行测试。

abstract	get	prototype
arguments	goto	RangeError
Array	has	ReferenceError
Boolean	include	RegExp
byte	Infinity	rounding
call	int	set
cast	internal	short
char	intrinsic	standard
Date	is	strict
decimal	isFinite	String
decodeURI	isNaN	synchronized
decodeURIComponent	JSON	SyntaxError
double	like	throws

dynamic	long	to
each	Math	transient
encodeURI	namespace	true
encodeURIComponent	NaN	type
Error	native	TypeError
eval	null	uint
EvalError	Number	undefined
false	Object	URIError
final	override	use
float	parseFloat	volatile
Function	parseInt	xml
generator	precision	

附录 C

CSS参考

这个附录列出 W3C 定义的 CSS 3 属性（w3.org/Style/CSS/currentwork）。

CSS 2 这个规范诞生于 20 世纪 90 年代中期，且在 1998 年 5 月就标准化了。此后不久人们就开始制定 CSS 2.1 规范，它的目标是使 CSS 2 更接近浏览器厂商已经实际实现的属性。跟其他规范一样，CSS 2.1 规范也是一拖再拖，直到 2011 年 6 月才完成标准化。

CSS 2 标准化后人们就开始制定 CSS 3 规范了，它的一大进步在于将原来的规范分成了不同的模块，这样每个模块都可独立发挥作用，不干扰其他模块。不过 CSS 3 也可谓历经磨难，正式发布的模块也就寥寥几个。

不过，也有好消息。浏览器制造商正竭尽全力在各自的浏览器中实现 CSS 3，因此目前大部分 CSS 3 属性实际上都已经能用了。本附录包括 CSS 3 中的绝大部分属性（不全）。最终采纳的属性会随着时间而改变，我们认为学习属性/浏览器/版本组合的最佳方式是尽量关注 caniuse.com/#cats=CSS。

因为本书是关于 JavaScript 的，我们只是稍微接触了 CSS 的皮毛。如果你想了解更多信息，我们推荐 Tom Negrino 和 Dori Smith 著的 *Styling Web Pages with CSS: Visual QuickProject Guide*（注意，就是我们两个写的书哦）。

表C-1　基本概念

属　性　名	值
HTML中的标签	`Link`
	`<style>...</style>`
	`< x style="声明;">`
分组	`x, y, z {声明;}`
上下文选择器	`x y z {声明;}`
类选择器	`.class`
ID选择器	`#id`
@规则	`@import`
	`@media`
	`@page`
	`@font-face`
重要性	`!important`

表C-2　伪元素和伪类

属　性　名	值
后一个元素	`:after`
锚元素	`a:active`
	`a:focus`
	`a:hover`
	`a:link`
	`a:visited`
前一个元素	`:before`
第一个元素	`:first`
第一个子元素	`:first-child`
左元素	`:left`
段落	`p:first-letter`
	`p:first-line`
右元素	`:right`

表C-3　页面属性

属　性　名	值
orphans	<整数>
page-break-after	auto
	always
	avoid
	left
	right
page-break-before	auto
	always
	avoid
	left
	right
page-break-inside	avoid
widows	auto
	<整数>

表C-4　用户界面属性

属　性　名	值
box-sizing	content-box
	padding-box
	border-box
cursor	<url>
	auto
	crosshair
	default
	pointer
	move
	e-resize
	ne-resize
	nw-resize
	n-resize
	se-resize
	sw-resize
	s-resize
	w-resize
	ew-resize
	ns-resize
	nesw-resize
	nwse-resize
	col-resize
	row-resize
	all-scroll
	zoom-in
	zoom-out
	text
	wait
	cell
	help
	progress
	context-menu
	vertical-text
	alias
	copy
	no-drop
	not-allowed
	none
outline	<轮廓颜色>
	<轮廓样式>
	<轮廓宽度>
outline-color	<颜色>
	invert
outline-style	<边框样式>
	auto
outline-width	<边框宽度>
resize	none
	both
	Horizontal
	vertical
text-overflow	clip
	ellipsis
	<string>

表C-5　视觉效果属性

属　性　名	值
max-lines	none
	<整数>
overflow	<overflow-x>
	<overflow-y>
overflow-x	visible
	hidden
	scroll
	auto
	paged-x
	paged-y

（续）

属　性　名	值
	paged-x-controls paged-y-controls fragments
overflow-y	visible hidden scroll auto paged-x paged-y paged-x-controls paged-y-controls Fragments

表C-6　背景和框属性

属　性　名	值
background	<background-color> <background-image> <background-position>/ <background-size> <background-repeat> <background-attachment> <background-clip> <background-origin>
background-attachment	scroll fixed local
background-clip	border-box padding-box content-box
background-color	<颜色>
background-image	<url> none
background-origin	border-box padding-box content-box
background-position	<百分数> <长度> top center bottom left right
background-repeat	repeat repeat-x repeat-y no-repeat space round
background-size	<百分数> <长度> contain cover auto
border	<border-width> <border-style> <border-color>
border-bottom	<border-bottom-width> <border-bottom-style> <border-bottom-color>
border-bottom-color	<边框颜色>
border-bottom-right-radius	<以长度值或者百分数值表示的边框圆角>
border-bottom-left-radius	<以长度值或者百分数值表示的边框圆角>
border-bottom-style	<边框样式>
border-bottom-width	<边框宽度>
border-color	<颜色>
border-image	<border-image-source> <border-image-slice> <border-image-width> <border-image-outset> <border-image-repeat>
border-image-outset	<长度> <数值>
border-image-repeat	Stretch repeat round space
border-image-slice	<数值> <百分数>
border-image-source	none <url>
border-image-width	<长度> <百分数> <数值> auto

（续）

属　性　名	值
border-left	<border-left-width> <border-left-style> <border-left-color>
border-left-color	<边框颜色>
border-left-style	<边框样式>
border-left-width	<边框宽度>
border-radius	<长度> <百分数>
border-right	<border-right-width> <border-right-style> <border-right-color>
border-right-color	<边框颜色>
border-right-style	<边框样式>
border-right-width	<边框宽度>
border-style	none hidden dotted dashed solid double groove ridge inset outset
border-top	<border-top-width> <border-top-style> <border-top-color>
border-top-color	<边框颜色>
border-top-left-radius	<以长度值或者百分数值表示的边框圆角>
border-top-right-radius	<以长度值或者百分数值表示的边框圆角>
border-top-style	<边框样式>
border-top-width	<边框宽度>
border-width	<长度> thin medium thick
box-shadow	none <长度> <颜色> inset

表C-7　表格属性

属　性　名	值
caption-side	top bottom
table-layout	auto fixed
border-collapse	collapse separate
border-spacing	<长度>
empty-cells	show hide
border-style	none hidden dotted dashed solid double groove ridge inset outset
vertical-align	baseline sub super top text-top middle bottom text-bottom <百分数> <长度>

表C-8　字体属性

属　性　名	值
font	<font-style> <font-variant> <font-weight> <font-stretch> <font-size>/<line-height> <font-family> caption icon menu message-box small-caption status-bar

（续）

属　性　名	值
font-family	<字体系列名> cursive fantasy monospace sans-serif serif
font-kerning	auto normal none
font-size	<绝对大小>（xx-small-xx-large） <相对大小>（smaller-larger） <长度> <百分数>
font-size-adjust	none number
font-stretch	normal ultra-condensed extra-condensed condensed semi-condensed semi-expanded expanded extra-expanded ultra-expanded
font-style	normal italic oblique
font-synthesis	none <粗细> <样式>
font-variant	none ruby <font-variant-caps> <font-variant-ligatures> <font-variant-numeric>
font-variant-caps	normal small-caps all-small-caps petite-caps all-petite-caps unicase titling-caps

（续）

属　性　名	值
font-variant-ligatures	normal none common-ligatures no-common-ligatures discretionary-ligatures no-discretionary-ligatures historical-ligatures no-historical-ligatures contextual no-contextual
font-variant-numeric	normal lining-nums oldstyle-nums proportional-nums tabular-nums diagonal-fractions stacked-fractions ordinal slashed-zero
font-variant-position	normal sub super
font-weight	normal bold 100-900
src	<url>

表C-9　单位

属　性　名	值
角度单位	deg grad rad turn
图像	<url>
长度单位	ch em ex px in cm mm pt pc rem vh vmax vmin vw

（续）

属　性　名	值
分辨率	dpi dpcm dppx
时间	s ms
URL	<url>

表C-10　文本属性

属　性　名	值
hanging-punctuation	none first force-end allow-end last
hyphens	none manual auto
letter-spacing	normal <长度>
line-break	auto loose normal strict
overflow-wrap	normal break-word
tab-size	<整数> <长度>
text-align	left right center justify start end match-parent
text-align-last	auto start end left right center justify
text-decoration	<text-decoration-line> <text-decoration-color> <text-decoration-style>
text-decoration-color	<颜色>
text-decoration-line	none underline overline line-through blink
text-decoration-skip	none objects spaces ink edges box-decoration
text-decoration-style	solid double dotted dashed wavy
text-emphasis-color	<颜色>
text-emphasis-position	over under right left
text-emphasis-style	none filled open dot circle double-circle triangle sesame <string>
text-indent	<长度> <百分数> each-line hanging
text-justify	auto none inter-word distribute
text-shadow	none <长度> <颜色>
text-transform	capitalize uppercase lowercase full-width none

（续）

属　性　名	值
text-underline-position	auto under left right
white-space	normal pre nowrap pre-wrap pre-line
word-break	normal keep-all break-all
word-spacing	normal <长度> <百分数>
word-wrap	normal break-word

表C-11　盒属性

属　性　名	值
clear	none left right both
float	left right none
margin	<长度> <百分数>
margin-bottom	<长度> <百分数>
margin-left	<长度> <百分数>
margin-right	<长度> <百分数>
margin-top	<长度> <百分数>
width	<长度> <百分数> auto
min-width	<长度> <百分数>
max-width	<长度> <百分数> none
height	<长度> <百分数> auto
min-height	<长度> <百分数>
max-height	<长度> <百分数> none
overflow	visible hidden scroll auto no-display no-content
overflow-style	auto scrollbar panner move marquee
overflow-x	visible hidden scroll auto no-display no-content
overflow-y	visible hidden scroll auto no-display no-content
padding	<长度> <百分数>
padding-bottom	<长度> <百分数>
padding-left	<长度> <百分数>
padding-right	<长度> <百分数>
padding-top	<长度> <百分数>
rotation	<角度>
visibility	collapse visible hidden

表C-12　视觉格式化属性

属 性 名	值
display	block inline inline-block list-item table inline-table table-row-group table-header-group table-footer-group table-row table-column-group table-column table-cell table-caption none
left	auto <长度> <百分数>
right	auto <长度> <百分数>
top	auto <长度> <百分数>
bottom	auto <长度> <百分数>
float	left right none
clear	none left right both
direction	ltr rtl
unicode-bidi	normal embed bidi-override
line-height	normal <数值> <长度> <百分数> none

（续）

属 性 名	值
position	static absolute relative fixed
z-index	auto <整数>

表C-13　列表属性

属 性 名	值
counter-reset	<标识符><整数> none
counter-set	<标识符><整数> none
counter-increment	<标识符><整数> none
list-style	<list-style-type> <list-style-position> <list-style-image>
list-style-image	<image> none
list-style-position	inside outside
list-style-type	disc circle square disclosure-open disclosure-closed decimal decimal-leading-zero hebrew lower-roman upper-roman lower-greek lower-alpha lower-latin upper-alpha upper-latin armenian georgian <string> none

表C-14　生成的内容属性

属　性　名	值
content	<字符串>
	<url>
	<标识符>
	<计数器>
	inhibit
	open-quote
	close-quote
	no-open-quote
	no-close-quote
	none
	normal
counter-increment	<标识符>
	<整数>
	none
counter-reset	<标识符>
	<整数>
	none
quotes	<字符串>
	none

表C-15　颜色属性

属　性　名	值
color	#000
	#000000
	hsl(HHH,S%,L%)
	rgb(RRR,GGG,BBB)
	rgb(R%,G%,B%)
	rgba(RRR,GGG,BBB,<alphavalue>)
	rgba(R%,G%,B%,<alphavalue>)
	transparent
	currentColor
	<关键词>
opacity	<alphavalue>

附录 D

其他学习资源

学完本书之后，你应该会急于开始用 JavaScript 为 Web 站点增色添彩。但是，关于 JavaScript 语言还有许多要学习的东西，而且你在编写代码时可能会遇到问题。

正如你所料，为这些问题寻找答案的最佳地方是在网上。在网上有许多资源可以帮助你解决问题以及深入了解 JavaScript。

在这个附录中，我们将介绍几个最有帮助的面向 JavaScript 的 Web 站点，还要提到本书作者认为有帮助的其他书。

但是，首先提醒你注意一点：网络不是静态不变的。Web 站点可能会改变它们页面的地址，所以随着时间的推移，我们列出的 URL 到你使用它们时可能会失效。我们仅仅是推荐这些 URL，但不保证它们一直有效。有时候，甚至整个站点都会消失。如果你发现一个链接失效了，那么请访问我们的配套 Web 站点（javascriptworld.com），看看我们是否发布了你所寻找的页面的新位置。

D.1 在网上寻找帮助

在 Mozilla 的网站上可以找到原始的 JavaScript 文档，但是在微软和独立 JavaScript 页面上有许多好信息。下面是一些最好的 JavaScript 网站。

D.1.1 浏览器厂商

既然是 Netscape 开发了 JavaScript，而 Mozilla 项目开发了开放源码的 Mozilla 和 Firefox（它们是 Netscape 浏览器的后继者），所以 Mozilla 项目提供了关于 JavaScript 语言和进一步开发的许多好信息。

1. JavaScript 中心

developer.mozilla.org/en/JavaScript

这个网站对所有水平的 JavaScript 用户都有帮助，其中包含工具和文档的链接（见图 D-1）。文档包括 Core JavaScript Reference（讨论 JavaScript 1.5），它介绍了 JavaScript 语言的基本知识，定义并解释了 JavaScript 中使用的概念。这个网站有一定的学习难度，示例也比较粗略，但是在消化了本书的内容之后，你应该能够理解这些资料。在这里还可以找到 Core JavaScript Guide，以及对 JavaScript 1.6 之后新增特性的解释。



图 D-1　Developer Center 的 Mozilla JavaScript 部分是进一步了解 JavaScript 的好地方

2. Venkman 调试器

developer.mozilla.org/en/venkman

当需要对 JavaScript 进行调试时，使用出色的调试工具肯定会对你有帮助。Venkman 是 Mozilla 项目提供的 JavaScript 调试器，它是一种适用于 Firefox、Thunderbird 和 Mozilla 的好工具，允许用户分步执行代码，设置断点，在脚本执行时检查对象和变量，以及处理 JavaScript 源代码。

3. Mozilla Hacks——Web 开发者博客

hacks.mozilla.org

该 blog 自称它“面向使用 Mozilla Firefox 和开放 Web 的人，突出前沿性的东西”——它的确如此。

4. Microsoft 的 JScript 语言

msdn.microsoft.com/hbxc2t98.aspx

JScript 是 Microsoft 自己的 JavaScript 版本，在 Microsoft 开发者网站上有自己的独立页面，如图 D-2 所示，登录网站可以学习 JScript 与 Mozilla 的 JavaScript 之间的相同点和不同点，还可以找到详细的 JScript 语言参考书和 JScript 用户手册。



图 D-2　JScript 是 Microsoft 开发的 JavaScript 变种

5. Surfin 的 Safari

webkit.org/blog/

虽然这个网站（如图 D-3 所示）不仅仅介绍 JavaScript，但它全面介绍了苹果的 Safari 浏览器及其 WebKit 渲染引擎——已不只是 Mac 才能用了。在这里可以直接从苹果的员工那得到关于 Safari 的下一个版本中包含什么功能以及不包含什么功能的信息。同时还可以找到 WebKit 可下载的 nightly 版本，让我们在苹果正式发布很久之前就能提前使用到新功能。

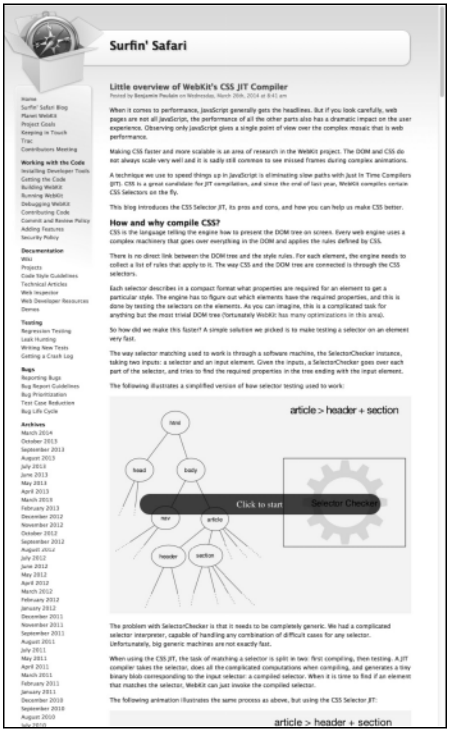


图 D-3　要想了解 Safari 的最新信息（Mac、Windows 或者 iOS 平台），登录 Surfin 的 Safari

D.1.2　博客及其他

有许多专注于 JavaScript 的博客。下面是我们喜欢的一些博客，但是还有许许多多出色的博客。

1. DailyJS

dailyjs.com

关于 JavaScript 的博客浮浮沉沉，但 DailyJS 一直陪伴大家。JavaScript 相关的新鲜事和链接都能从这个博客中找到。

2. QuirksMode

quirksmode.org/blog/

Peter-Paul Koch 是荷兰的一位 JavaScript 开发人员。他的站点（见图 D-4）最出色的特色是，它非常及时地提供关于浏览器及其 JavaScript 功能的最新消息。它并非一个指南性站点，但是提供了许多

对初级脚本开发人员有帮助的基本信息。



图 D-4　Peter-Paul Koch 的 QuirksMode 站点常常会在其他人之前提供最新的 JavaScript 信息

D.2　离线资源

图书

尽管本书的作者希望光靠本书就能够让你成为 JavaScript 专家，但是我们知道这是不现实的，在你消化了本书之后，可能还需要了解更多信息。图书市场上有数不清的 JavaScript 图书，下面是我们认为最好的几本书（排名不分先后）。

1.《JavaScript 权威指南》

由 David Flanagan 执笔，O’Reilly 出版。这是一本针对 JavaScript 语言全面的参考手册。“胆小勿入”，很多专家通过本书查阅生僻的操作符并制定怪异的语法。在经过了漫长的等待之后，本书的第 6 版出版了，带来了最新的知识库。

2.《ppk 谈 JavaScript》

Peter-Paul Koch 是公认的 JavaScript 大师之一。在 New Riders 出版的这本书中，他使用为客户创建的真实的脚本示例帮助读者在理论和实践两方面掌握 JavaScript。

3.《精通 JavaScript》

由 John Resig（jQuery 名人）执笔，辅助中级 JavaScript 开发人员成为高级开发人员。本书不适合新手，它是一本学完后值得深入研读的好书。

D.3　解决问题的技巧

对于编程新手来说，可能会感觉每行都有错误，不过庆幸的是，有很多在线资源可以帮我们度过这个困难重重的时期。下面是精选的一些资源。

D.3.1　缺失的插件

1. Firebug 调试器

getfirebug.com

不管你是否喜欢 Mozilla 的 Venkman，都会爱上 Firebug，如图 D-5 所示。在 Firefox 中使用 Firebug 不仅可以调试脚本，还可以调试 HTML 和 CSS。Firebug 还有以下优点：免费、文档良好、记录便捷、可随意设置断点和 DOM 支持。

如果说我们的编程生活中唯一的遗憾就是在 IE、Safari 和 Opera 中缺失类似功能的话，请登录 http://getfirebug.com/lite.html，试试 Firebug Lite，如图 D-6 所示。虽然功能方面不像 Firebug 那么全面，但跟踪 bug 时，它提供的简单通用界面是非常方便的。

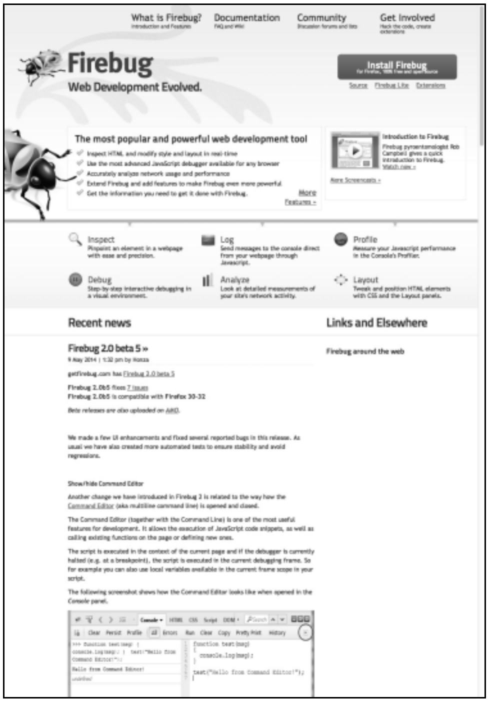


图 D-5　作为 Firefox 的插件，Firebug 提供了相当强大的控制功能和多样化的代码检查方式

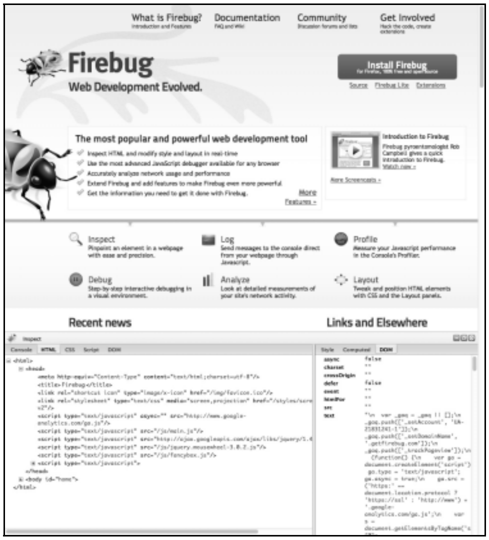


图 D-6　如果我们使用的不是 Firefox，那么也可以运行 Firebug Lite。图中显示的是图 D-5 中的页面在 Safari 中的显示效果

2. JSHint

jshint.com

如果你跟我们一样，在遇到 HTML 和 CSS 的问题时第一反应就是去找一个验证器，而同时又希望在 JavaScript 方面也有类似的工具的话，那你可能听说过 JSLint。JSLint 是一个代码检查工具。再如果，你还跟我们（以及其他很多的 JavaScript 程序员）一样，发现 JSLint 实际用起来不太好用，那么 JSHint 就是你需要的。JSHint 的设计初衷不仅是代码检查，而且富有灵活性，不会像 JSLint 那样出现“说一不二”的死板规定。

3. Can I Use...

caniuse.com/#cats=JS_API

如果你编写比较前沿的 JavaScript 代码，可能会遇上浏览器不兼容问题。一旦出现这类问题，你可以来这个网站看看，这里详细列出了哪种浏览器支持哪些功能、不支持哪些功能，这些功能肯定涵盖你正绞尽脑汁使用的某几种。

D.3.2　在线pastebin

你是否曾经希望不仅仅是拿浏览器来测试代码，同时还想编写代码？是啊，为什么我们必须要来回切换界面？pastebin 的功能之一就是在同一窗口中编写和运行代码。

另一个方便的功能是可以保存文件并展现给其他人（也就是在你需要帮助的时候指给别人看）。而更酷的是它可以修改你的代码，帮你查找和修正那些令人讨厌的 bug。

网上有很多关于 pastebin 的网站，一些是支持 HTML、CSS 和 JavaScript 的。我们（及一部分人）喜欢下面两个。

1. JSBin

jsbin.com

2. JSFiddle

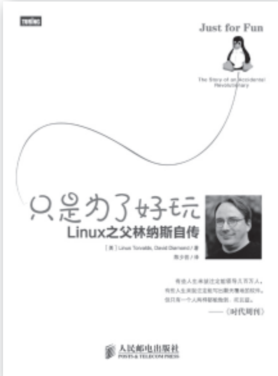
jsfiddle.net

TURING
图灵教育

更多推荐



HTML5 与 CSS3 基础教程（第 8 版）
作者：Elizabeth Castro,Bruce Hyslop
书号：978-7-115-35065-7
定价：69.00 元



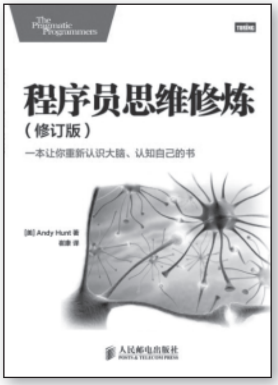
只是为了好玩：Linux 之父林纳斯自传
作者：Linus Torvalds，David Diamond
书号：978-7-115-36164-6
定价：49.00 元



HTML5 与 CSS3 实例教程（第 2 版）
作者：Brian P. Hogan
书号：978-7-115-36340-4
定价：49.00 元



高效程序员的 45 个习惯：敏捷开发修炼之道（修订版）
作者：Venkat Subramaniam
Andy Hunt
书号：978-7-115-37036-5
定价：45.00 元



程序员思维修炼（修订版）
作者：Andy Hunt
书号：978-7-115-37493-6
定价：49.00 元



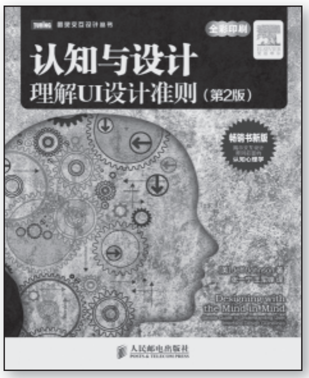
我们要自学
作者：张玳
书号：978-7-115-37438-7
定价：39.00 元



如何变得有思想：阮一峰博客文集
作者：阮一峰
书号：978-7-115-37364-9
定价：49.00 元



程序员健康指南
作者：Joe Kutner
书号：978-7-115-36716-7
定价：39.00 元



认知与设计：理解 UI 设计准则（第 2 版）
作者：Jeff Johnson
书号：978-7-115-36410-4
定价：69.00 元

（第9版）

JavaScript基础教程

“学习JavaScript必备书目！”

——JavaScript.about.com

“本书是我所见过的讲解最清晰、内容最全面、最吸引人的JavaScript程序设计图书。”

——亚马逊读者评论

在Web开发如火如荼的今天，JavaScript理所当然地成为广大开发人员必须熟练掌握的一项主流技术。根据有关编程语言市场份额的调查，JavaScript已经成为Web开发人员使用最多的开发语言。

本书被奉为经典JavaScript入门书，以易学便查、图文并茂、循序渐进而著称，作者善于用常见任务讲解语言知识。书中除了讲述JavaScript编程的必知必会知识外，还同时兼顾了DOM、XML、Ajax、jQuery等重要的技术内容。多年来，本书不断重印改版，原版累计销售已经近20万册。第7版和第8版的中文版出版后也多次重印，广受国内读者好评。第9版对上一版进行了全面修订和更新，更新了针对各主要浏览器的代码并增加了一章全新内容，介绍为移动设备编写脚本的方法。

通过本书，读者可以迅速轻松掌握用JavaScript进行Web开发的基本技能，并了解最佳实践，领悟其中真谛。

Peachpit Press

图灵社区：iTuring.cn

热线：(010)51095186转600

分类建议 计算机/程序设计/JavaScript

人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-38522-2

9 787115 385222

ISBN 978-7-115-38522-2

定价：69.00元